MODÈLE D'AGENT

POUR LA

SIMULATION RÉPARTIE

PAR ÉVÉNEMENTS DISCRETS

THÈSE

présentée à la Faculté des sciences, pour obtenir

le grade de Docteur ès sciences, par

Pontien DÉGUÉNON

UNIVERSITÉ DE NEUCHÂTEL
Institut d'Informatique et
d'Intelligence Artificielle
rue Émile-Argand 11
2007 Neuchâtel, Suisse

IMPRIMATUR POUR LA THÈSE

Modèle d'agent pour la simulation répartie par événements discrets.

de M. Pontien Déguénon

UNIVERSITÉ DE NEUCHÂTEL

FACULTÉ DES SCIENCES

La Faculté des sciences de l'Université de Neuchâtel sur le rapport des membres du jury,

Messieurs P.-J. Erard, H.-H. Nägeli, J. Savoy, J. Pasquier (Fribourg) et A. Drogoul (Paris)

autorise l'impression de la présente thèse.

Neuchâtel, le 4 juillet 1996

Le doyen:

R. Dändliker

Si tu te tais

tais-toi par amour

Si tu parles

parle par amour

Si tu corriges

corrige par amour

Si tu pardonnes

pardonne par amour

aie au fond du cœur

la racine de l'amour

St-Augustin

Je dédie ce travail
aux parents
qui œuvrent pour la
réussite de chacun de leurs
enfants

Remerciements

Je souhaite exprimer ma gratitude à celles et à ceux qui ont contribué à la réalisation de cette thèse, en particulier:

- Pierre-Jean Erard, mon directeur de thèse, pour ses nombreuses idées enrichissantes, les discussions fructueuses, son encouragement et son soutien scientifique.
- Hans-Heinrich Nägeli pour avoir accepté de participer à ce jury, son encouragement et ses critiques constructives suite à la lecture du manuscrit.
- Jacques Pasquier pour avoir accepté de participer à ce jury, la lecture attentive du manuscrit et ses précieuses suggestions.
- Alexis Drogoul pour avoir accepté de participer à ce jury et pour la sympathie dont il a fait preuve lors des colloques.
- Jacques Savoy pour avoir accepté de participer à ce jury.
- Olivier von Dach que j'associe à la réussite de cette thèse, pour son travail méticuleux et les longues discussions bénéfiques à la réalisation de l'environnement de programmation.
- Les membres de l'Institut d'Informatque et Intelligence Artificielle.
- Roger Bellet pour la lecture attentionnée et fructueuse de cette thèse et pour son soutien moral.
- Ma famille qui a su me passionner pour les études, m'encourager et m'entourer de leur affection; ainsi que mes amis, ils savent ce que je leur dois.

Enfin à mon épouse, Marianne, pour sa compréhension, sa persévérance et sa patience; ce travail est également le sien.

Table des matières

1 Introduction générale **3**

1.1 Définition et classification 3

1.2 Simulation par événements discrets (SED) 4

1.3 Simulation parallèle et simulation répartie 4

1.4 Pourquoi la simulation répartie? 5

1.5 Concept de la simulation répartie 6

1.5.1 Synchronisation 6

1.5.2 Simulation répartie sûre et risquée 6

1.5.3 Limite des protocoles de synchronisation existants 6

1.6 Thèse défendue 7

1.7 Plan de la thèse 7

I Simulation répartie **11**

2 Introduction à SED **13**

2.1 Introduction 13

2.2 Caractéristiques du système et modèle simulé 14

2.2.1 Principes du système physique 14

2.2.2 Propriétés des modèles exécutables 14

2.3 Programme de simulation 15

2.3.1 Simulateur correct 16

2.3.2 Différents types de messages 16

2.3.3 Mécanisme d'avancement du temps simulé 17

2.4 Simulation séquentielle 18

2.5 Paradigme de processus logiques 20

2.5.1 Problème de terminaison instantanée 21

2.5.2 Encapsulation de *PP*s par des *PL*s 21
2.5.3 Analyse du paradigme de *PL*s 21
2.6 Parallélisation de la simulation 22
2.6.1 Architecture cible des machines 22
2.6.2 Accélération matérielle 23
2.6.3 Décomposition temporelle 23
2.6.4 Décomposition spatiale 24
2.6.5 Caractéristiques de la simulation répartie 25
2.7 Contexte distribué 26
2.8 Relation et contrainte de causalité 26
2.8.1 Relation de causalité entre événements 26
2.8.2 Contrainte de causalité 27
2.9 Conclusion 28

3 Simulation sûre 29
3.1 Opérations d'un *PL* 29
3.2 Problème d'impasse 30
3.3 Résolution de l'impasse 32
3.3.1 Prévention des impasses 32
3.3.2 Détection-Guérison des impasses 34
3.4 Autres approches de résolution de l'impasse 35
3.5 Analyse des mécanismes 36

4 Simulation optimiste 37
4.1 Paradigme de temps virtuel 38
4.2 Origine de la "distorsion temporelle" 39
4.2.1 Domaines d'application de la distorsion temporelle 39
4.2.2 Problème de synchronisation des temps virtuels 39
4.3 Mécanisme de distorsion temporelle 40
4.4 Contrôle local du simulateur 41
4.4.1 Description d'un *PL* 41
4.4.2 Message et Antimessage 42
4.4.3 Rollback et mécanisme d'annulation 43
4.4.4 Retour au dernier état cohérent du système 44
4.5 Contrôle global du simulateur 47

4.6 Conclusion 48

5 Comparaison des approches de simulation **49**

5.1 Comparaison 49

5.1.1 Différences dans le paradigme de simulation 49

5.1.2 Restrictions statiques 50

5.1.3 Mécanismes de synchronisation 51

5.1.4 Étude de la performance:force et faiblesse 51

5.2 Conclusion 52

II Etude du contrôle local de la simulation optimiste **53**

6 Optimisation de la distorsion temporelle **55**

6.1 Différents mécanismes de rollback 56

6.1.1 Annulation agressive 56

6.1.2 Annulation paresseuse 56

6.1.3 Réévaluation paresseuse 56

6.1.4 Méthodes de prévision 57

6.1.5 Chasse aux calculs erronés ("Wolf calls") 57

6.1.6 Annulation directe 57

6.1.7 Relaxation des rollbacks 57

6.2 Techniques de récupération de mémoire 58

6.3 Sauvegarde des états 59

6.4 Protocoles pour limiter l'optimisme 60

6.4.1 Méthode de la fenêtre temporelle 60

6.4.2 Protocole d'ordonnancement 61

6.5 Protocole de stabilisation 61

6.6 Synthèse des travaux précédents 63

6.7 Orientation de la recherche 64

7 Contrôle dynamique optimiste **65**

7.1 Introduction 66

7.2 Objectif de notre approche 66

7.3 Théorie du contrôle 67

7.3.1 Contrôle de système ouvert 68

7.3.2 Contrôle de système fermé 68
7.3.3 Systèmes adaptatifs 68
7.4 Fondement du contrôle dynamique d'un *PL* 71
7.4.1 Concept de qualité de synchronisation 73
7.4.2 Indice de performance:"qualité de synchronisation" 74
7.4.3 Une approche de Modèle optimal 74
7.5 Conclusion 75

8 Réalisation du contrôle dynamique 77
8.1 Introduction 77
8.2 Sauvegarde des états 78
8.2.1 Sauvegarde incrémentale d'états 79
8.2.2 Sauvegarde périodique d'états 79
8.2.3 Optimisation de la période de sauvegarde 80
8.2.4 Algorithmes adaptatifs de la sauvegarde d'états 81
8.2.5 Empirique pour la période de sauvegarde d'états 81
8.3 Taille adaptative de la fenêtre temporelle 82
8.3.1 Taille statique des fenêtres temporelles 83
8.3.2 Ajustement dynamique de la fenêtre temporelle 83
8.4 Ordonnancement des *PLs* 85
8.4.1 Critères d'un algorithme d'ordonnancement 85
8.4.2 Algorithmes d'ordonnancement de *PLs* 85
8.4.3 Analyse des algorithmes 86
8.4.4 Ordonnancement selon la "qualité de synchronisation" 87
8.5 Conclusion 88

III Etude du contrôle global de la simulation optimiste 91

9 Détermination de GVT 93
9.1 Introduction 93
9.2 Etat global cohérent 94
9.2.1 Caractéristique d'un état global cohérent 94
9.3 Que désigne-t-on par *GVT*? 95
9.3.1 *GVT* valeur abstraite et opérationnelle 95
9.3.2 Critères ou qualité de *GVT* 97

9.3.3 Importance de *GVT* 98
9.3.4 Difficulté d'estimation de *GVT* 99
9.4 Problèmes des messages en transit 99
9.4.1 Propriétés PAPS de canaux 100
9.4.2 Messages avec accusé de réception 100
9.4.3 L'approche de Samadi 101
9.5 Algorithmes à contrôle centralisé 101
9.5.1 Principe de fonctionnement 101
9.5.2 Variantes de l'algorithme à contrôle centralisé 102
9.5.3 Analyse des algorithmes à contrôle centralisé 103
9.6 Algorithmes à structure hiérarchique 103
9.6.1 Algorithmes à structure de graphe 103
9.6.2 Algorithmes en anneau 105
9.7 Algorithme à contrôle réparti 107
9.8 Critique des algorithmes de *GVT* 108

10 Algorithme LT:GVT local **109**
10.1 Introduction 109
10.2 Critique des algorithmes de GVT existants 110
10.2.1 Inconvénients des approches existantes 111
10.2.2 Motivation 111
10.3 Que désigne-t-on par *LT*? 112
10.4 Modèle asynchrone de simulation 114
10.4.1 Hypothèses 115
10.4.2 Solution conceptuelle 116
10.4.3 Absence d'impasse 117
10.5 Fonctionnement de l'algorithme *LT* 118
10.5.1 Phase de pré-traitement 118
10.5.2 Phase de traitement 119
10.6 Formulation de l'avancement de *LT* 121
10.7 Algorithmes de traitement des jetons 122
10.8 Originalité de notre approche 124
10.9 Conclusion 124

11 L'exactitude de l'algorithme LT **127**

11.1 Notations des messages à traiter 127
11.1.1 Messages non-traités dans un PL 127
11.1.2 Messages non-traités en transit 128
11.1.3 Messages en attente de traitement 128
11.1.4 Date locale sûre 129
11.1.5 Estimation de la date locale sûre 129
11.2 Exactitude partielle 130
11.2.1 Preuve 130
11.3 Exactitude totale 135
11.4 Conclusion 136

IV Simulation multi-agents et simulation par événements discrets 139

12 Un Modèle d'agent pour la SED 141
12.1 Relaxation restreinte et simulation optimiste 141
12.2 Agent:composant individuel 142
12.2.1 Représentation d'un agent AgSED 142
12.2.2 Architecture d'un agent AgSED 144
12.3 Modèle social de l'agent AgSED 147
12.3.1 Coopération 147
12.3.2 Organisation 147
12.3.3 Apprentissage et révision des croyances 148
12.4 Résolution de conflits de causalité 148
12.5 Conclusion 149

13 SED multi-agents 151
13.1 Simulation d'un système réel 151
13.2 Simulateur Multi-Agents pour SED 153
13.2.1 Différents types d'agents 153
13.2.2 Mesures effectuées sur le simulateur 156
13.2.3 Comment spécifier un agent logique 156
13.2.4 Actions d'un agent logique 157
13.3 Critères de la conception de SEDMA 157
13.4 Réalisation de SEDMA en Smalltalk 158
13.4.1 Moteur de SEDMA 159

13.4.2 Différentes couches de SEDMA 162
13.4.3 Collection des données statistiques 164
13.5 Terminaison de la simulation 165
13.6 Conclusion 166

14 Evaluation de SEDMA **167**
14.1 Simulation d'un système de file d'attente 168
14.1.1 Cheminement des clients 168
14.1.2 Nœud source de simulation 168
14.1.3 Nœud puits de simulation 169
14.1.4 Nœud branchement de simulation 169
14.1.5 Nœud station-service 170
14.2 Structure de la simulation d'un nœud 173
14.2.1 Structure élémentaire d'un nœud 174
14.3 Analyse quantitative 175
14.3.1 Système élémentaire 176
14.3.2 Système composé 177
14.4 Spécification des nœuds et leur analyse 178
14.4.1 Nœud source 179
14.4.2 Nœud branchement 179
14.4.3 Nœud station-service 179
14.4.4 Nœud puits 181
14.5 Etude de la performance 181
14.5.1 Cadre d'expérimentation 181
14.5.2 Description des modèles simples 182
14.5.3 Description des modèles complexes 185
14.5.4 Analyse des résultats 186
14.5.5 Synthèse des résultats selon les différentes topologies 195

15 Conclusion générale **197**
15.1 Contribution au contrôle local 198
15.2 Contribution au contrôle global 198
15.3 Perspectives 199
15.3.1 Répartir le simulateur 199
15.3.2 Améliorer l'interface 199

15.3.3 Analyse statistique avancée 200

A Environnement interactif et graphique de SEDMA **201**

B Exemple de spécification du système **205**

C Exemple d'analyse des résultats **207**

Liste des figures

1.1 Structure hiérarchique des chapitres 8

2.1 Simulateur séquentiel 18
2.2 *Structure algorithmique d'un simulateur séquentiel* 19
2.3 Structure d'un PL 20

3.1 Simulation répartie ne pouvant pas progresser (famine) 31
3.2 Simulation répartie dans un circuit 31
3.3 Réseau simulé pouvant conduire à une famine 35

4.1 Un processus de synchronisation de temps de virtuels 40
4.2 Description d'un PL optmiste 42
4.3 Mécanisme de synchronisation des messages 44
4.4 Description du rollback 46

5.1 Canal c' utilisé rarement 50

7.1 Contrôle de système ouvert 68
7.2 Contrôle de système fermé 69
7.3 Contrôle de système auto-adaptatif 69
7.4 Contrôle d'un modèle fermé de SAMR 70
7.5 Contrôle d'un modèle ouvert de SAMR 71
7.6 Système de PLs 72
7.7 Modèle de contrôle adaptatif d'un PL 72

8.1 Restitution des états sauvegardés pour $\chi = 4$ 80
8.2 *Algorithme de contrôle adaptatif de la période de sauvegarde d'états* 82
8.3 Fenêtre temporelle δt 83
8.4 *Algorithme de contrôle adaptatif de la taille de fenêtre temporelle* 84

8.5 *Algorithme d'auto-ordonnancement des PLs* 87

9.1 graphe équilibré et symétrique ("binary-tree-like forwarding") 104
9.2 *Structure du jeton circulant à un niveau* 106

10.1 Nœud en aval ou en amont de n 112
10.2 Deux graphes sans circuits G1 et G2 indépendants 112
10.3 Deux graphes G1 et G2 dépendants 113
10.4 Situations typiques d'impasse 117
10.5 *Traitement associé à la réception d'un jeton* 122
10.6 *Traitement du jeton rouge (se rendre à l'élément de tête)* 122
10.7 *Traitement du jeton vert (calcul de LT)* 123
10.8 *Traitement du jeton bleu (rapporter LT à l'élément de tête)* 123
10.9 *Traitement du jeton jaune (propagation de LT)* 124

12.1 Architecture d'un AgSED (agent logique) 144

13.1 Système réel et ses sous-systèmes 152
13.2 Différentes parties du simulateur 159
13.3 Composants d'un acteur **Actalk** 160
13.4 Interprétation d'un message actalk 161
13.5 Différentes couches de SEDMA 162
13.6 Comportement de remplacement 162

14.1 Cheminement statique et dynamique des clients 169
14.2 La loi hyper-exponentielle 172
14.3 un réseau de files d'attente, modèle d'un système 173
14.4 Simulateur élémentaire:réception-émission intégrée 174
14.5 Simulateur élémentaire:réception-émission séparée 174
14.6 Structure d'un système élémentaire 176
14.7 Structure d'un système composé 177
14.8 Différents algorithmes de la thèse 182
14.9 Réseau simulé du systéme de lavage de voitures 183
14.10Cheminement statique et dynamique des clients 184
14.11Topologie des modèles complexes simulés 185
14.12Topologie du modèle diffusion-fusion 186
14.13performance de l'algorithme optimiste - linéaire 187

14.14Performance de l'algorithme LT - linéaire 187
14.15Performance de l'algorithme adaptatif - linéaire 188
14.16Performance de l'algorithme optimiste - arbre 189
14.17Performance de l'algorithme LT - arbre 189
14.18Performance de l'algorithme adaptatif - arbre 190
14.19Performance de l'algorithme LT et adaptatif - arbre 190
14.20Performance de l'algorithme optimiste pour l'anneau 191
14.21Performance de l'algorithme LT pour l'anneau 191
14.22Performance de l'algorithme adaptatif pour l'anneau 192
14.23Performance de l'algorithme optimiste - atelier de production 192
14.24Performance de l'algorithme LT - atelier de production 193
14.25Performance de l'algorithme adaptatif - atelier de production 193
14.26Performance de l'algorithme adaptatif - diffusion-fusion 194
14.27Performance de l'algorithme LT - diffusion-fusion 194

A.1 Environnement interface de SEDMA 202

Chapitre 1

Introduction générale

1.1 Définition et classification

La simulation est une méthode utilisée pour étudier le comportement et pour analyser les performances d'un système réel (phénomène réel) existant ou en conception. Ces systèmes, souvent complexes, parfois inaccessibles et difficiles à contrôler, sont représentés par un ***modèle***. Ce dernier est une description plus ou moins détaillée du système réel en rapport avec le degré de finesse requis par l'analyse que l'on veut réaliser. Compte tenu des objectifs de la simulation, les paramètres du modèle peuvent être modifiés tout au long de l'étude. Grâce à cette flexibilité du modèle, la simulation analyse des situations complexes qu'on ne peut résoudre de manière analytique.

On distingue deux grandes classes de simulation: la simulation des systèmes continus et la simulation des systèmes à événements discrets.

Dans la simulation des systèmes continus, le modèle évolue d'une façon continue dans le temps et sur un espace d'état continu. Le modèle est représenté sous la forme d'équations différentielles ordinaires ou aux dérivées partielles où l'on étudie la variation temporelle continue des variables du modèle dans l'équation. Cette classe de simulation s'adapte bien à l'étude de vol d'un avion où l'évolution des variables d'état du système est suivie à chaque instant [LK91] [Fis95].

Dans la simulation des systèmes à événements discrets, le modèle évolue à des instants précis (*discrets*), souvent aléatoires lorsqu'un changement survient dans le système d'état: l'***événement***. Ce modèle est représenté sous la forme de règles de succession d'événements qui portent chacun leur date d'exécution. Deux sous-classes de simulation des systèmes à événements discrets se dégagent de la représentation des règles de succession:

- La simulation équationnelle, où la succession des événements est exprimée sous forme d'équations de récurrence. La progression des indices de récurrence simule l'évolution de la simulation. L'algèbre $(max, +)$ [BQ92] permet la simplification de ces équations et facilite ainsi l'étude de ces systèmes et la parallélisation de leur simulation [BC93].

- La simulation par événements discrets, où la succession des événements est définie

par un mécanisme de type cause à effet. Le traitement d'un événement à une date de simulation en engendre d'autres, de types différents ou identiques, et provoque un changement d'état du système. Cette sous-classe de simulation constitue l'objet de notre étude.

Cette thèse a trait à la simulation de systèmes à événements discrets qui tiennent compte des méthodes développées ces dernières années en architectures parallèles et réparties: la simulation parallèle et répartie.

1.2 Simulation par événements discrets (SED)

La simulation par événements discrets permet de modéliser un trés grand nombre d'applications allant des ateliers de production aux réseaux de communication et aux systèmes informatiques eux-mêmes. Dans certains cas, les problèmes modélisés sous forme de graphe d'événements [Can94] offrent une étude analytique qui permet de dégager une solution [Mol88] [CA93] et [Ho92]. Malheureusement, dans de nombreux cas, le modèle est trop complexe pour une étude analytique et seule la simulation du modèle permet d'obtenir une solution [Ler80].

La simulation classique ou séquentielle de la SED est basée sur la notion d'échéancier qui contient la suite des évènements à simuler. A cause de la complexité toujours croissante des modèles à simuler et de l'avènement des machines parallèles, on s'est intéressé, au début des années 80, à la parallélisation de la simulation afin d'améliorer les performances [PMW79] et [CHM79]. En effet, la parallélisation qui permet d'utiliser toutes les ressources de calcul du matériel cible (pour diminuer le temps de calcul) peut conduire soit à simuler plus rapidement certaines simulations – le moindre gain de temps peut se révéler profitable [WHF+89] – soit à réaliser des simulations qui n'auraient pu être élaborées autrement [Mis86] et [RW89b].

L'arrivée des machines parallèles à mémoire partagée ou à mémoire distribuée a permis d'effectuer des expérimentations jusqu'alors impossibles: c'est la simulation parallèle [Fuj88], [Fuj90a] et [RMM88]. L'arrivée sur le marché des machines parallèles – à des prix trés avantageux – mises en réseau a favorisé la mise en œuvre de nouveaux modèles de simulation partitionnés sur plusieurs processeurs [Mis86]: c'est ***l'exécution parallèle de la simulation***.

1.3 Simulation parallèle et simulation répartie

La *simulation parallèle*, à l'origine simulation répartie [CM79a], se rapporte à l'exécution parallèle de la simulation par événements discrets. Les premières recherches [Rey88] décrivent la coordination des processus coopératifs qui découle d'une simulation parallèle. Plus récemment, la simulation par événements discrets géographiquement répartie est assimilée à la simulation répartie. Sans vouloir établir une définition formelle, nous distinguons la simulation répartie de la simulation parallèle selon les critères du temps de communication entre les processus et la meilleure utilisation des processeurs mis à disposition.

Dans la *simulation répartie*, le délai de communication est de l'ordre de la milliseconde. Il sert à joindre les processeurs physiquement séparés pour leur permettre de coopérer afin d'atteindre l'objectif de la simulation. La simulation répartie ne met pas l'accent sur l'utilisation des processeurs ou le temps minimum pour traiter la simulation, bien que le temps réel nécessaire à l'accomplissement d'une tâche puisse les amener à traiter ces sujets.

La simulation parallèle, quant à elle, met l'accent sur l'utilisation optimale des processeurs et la minimisation de la période de simulation. Le délai de communication entre les processus est très petit. Il est au plus de l'ordre de la microseconde. A priori, la recherche de la simulation parallèle ne s'occupe pas des questions de temps réel.

Cependant, la nature de la connexion différencie les deux types de simulation. Dans la simulation parallèle, la communication entre les processeurs est très courte et conçue à l'intérieur de l'ordinateur. Il possède l'avantage d'un réseau homogène de processeurs et simplifie les problèmes conceptuels de synchronisation. Dans une simulation répartie, en revanche, le modèle est réparti sur un ensemble d'ordinateurs géographiquement répartis sur plusieurs sites. Le problème est que les canaux de communication sont utilisés à plusieurs fins au cours de la simulation: ils servent à synchroniser, à passer les messages d'événements et sont utilisés simultanément par plusieurs utilisateurs. Des problèmes supplémentaires de contrôle de flux et d'accès concurrentiel sur les canaux peuvent donc apparaître.

Bien plus, dans notre modèle, nous tenons compte de l'être humain dans le cycle du programme. Alors, l'interactivité de l'homme est intégrée dans le modèle pour opérer depuis un site sur l'ensemble du réseau géographiquement réparti. L'interactivité de l'homme augmente la complexité en ajoutant une contrainte supplémentaire: la réactivité du simulateur.

Nous ne distinguons pas les deux types de simulation (répartie et parallèle) dans la thèse car il y a de nombreux recoupements dans les diverses architectures de machines.

1.4 Pourquoi la simulation répartie?

Ces dernières années, la simulation répartie fut l'objet d'études de plusieurs chercheurs. Pourquoi cet intérêt des chercheurs?

D'un point de *vue pragmatique*, cet intérêt est lié à l'étendue des problèmes complexes que la simulation répartie couvre dans des domaines tels que l'ingénierie de modélisation, l'informatique, l'économie et le secteur militaire. Contrairement à la simulation séquentielle, la simulation répartie exécute simultanément plusieurs parties du problème complexe simulé et, par conséquent, accélère le temps total d'exécution de la simulation.

D'un point de *vue académique*, la simulation répartie est intéressante car elle traite les problèmes irréguliers difficiles à paralléliser dans la pratique [Liv85]. Une solution satisfaisante aux problèmes de simulation répartie nécessite une nouvelle approche de résolution où les calculs parallèles forment un tout pour aboutir à une solution. Très tôt, on a remarqué que les techniques de vectorisation de l'architecture des super-ordinateurs n'accélèrent guère la simulation à cause de la dépendance de données et des problèmes de

synchronisation.

Notre étude porte uniquement sur des modèles de simulation asynchrones où les événements surviennent à des intervalles de temps irréguliers. L'exécution parallèle des événements aux différents instants introduit des problèmes intéressants de synchronisation qui sont au cœur de tout problème de simulation répartie.

1.5 Concept de la simulation répartie

1.5.1 Synchronisation

Pour simuler l'évolution d'un phénomène réel dans le temps, on construit un modèle constitué de tâches actives qui interagissent. Une tâche active est simulée par un processus logique (PL) communiquant par échange de messages avec les autres PLs. Ces messages correspondent aux événements du système simulé; ils sont caractérisés par deux attributs, les actions associées à l'événement et la date de traitement de celui-ci. Dans la simulation répartie, les PLs sont placés sur des processeurs différents, voire des ordinateurs différents, géographiquement répartis et reliés entre eux par des cauaux de communication orientés. Chaque processus logique possède son horloge locale.

Dans l'évolution de la simulation, chaque PL attend la réception de messages d'événement pour mettre à jour son état local et son horloge locale. Les vitesses d'exécution des PLs étant différentes et le temps de transmission des messages d'un processus à un autre étant arbitraire mais fini, les horloges locales des différents PLs peuvent avoir des heures différentes. Dès lors, une ***erreur de causalité*** se produit lorsqu'un PL reçoit un message d'événement dont la date de traitement est antérieure à l'heure de son horloge. *Le problème de la simulation répartie est de définir un protocole de synchronisation qui garantisse que chaque PL traite chronologiquement les messages d'événement reçus.*

1.5.2 Simulation répartie sûre et risquée

Il existe deux protocoles de synchronisation: l'un *sûr* où l'erreur de causalité ne survient jamais, l'autre *risqué* ou optimiste au cours duquel l'erreur de causalité peut se produire mais où un mécanisme de retour arrière ou ***rollback*** permet de récupérer l'erreur.

1.5.3 Limite des protocoles de synchronisation existants

Nous avons étudié les deux stratégies de synchronisation pour identifier leurs qualités, leurs défauts et leurs limites, puis pour les comparer (*partie 1*). De cette analyse résulte le constat suivant: la stratégie de synchronisation sûre n'est pas robuste et elle ne permet pas d'exploiter au mieux le parallélisme inhérent au modèle simulé; pour sa part la stratégie de synchronisation risquée exploite pleinement le parallélisme, mais elle consomme beaucoup de mémoire et de temps de calcul.

1.6 Thèse défendue

Nous proposons dans cette thèse un modèle d'agent AgSED qui englobe les qualités des deux protocoles de synchronisation afin d'établir le meilleur compromis possible entre la simulation sûre et la simulation risquée (partie 4). Notre étude part de la stratégie de synchronisation optimiste, basée sur deux mécanismes de contrôle: le contrôle local et le contrôle global. Notre apport s'appliqu aux deux mécanismes de contrôle.

Nous avons remplacé le contrôle local par un contrôle local dynamique, où chaque PL évalue sa qualité de synchronisation par rapport à ses voisins. La qualité de synchronisation d'un PL est une fonction conçue de manière à réduire le nombre de rollbacks et à maximiser les messages d'événement ne provoquant pas d'erreur de causalité. Selon la valeur de cette fonction, on modifie dynamiquement la fréquence de sauvegarde d'états de chaque PL, sa priorité d'ordonnancement sur un processeur et sa fenêtre temporelle (période de sauvegarde) afin d'équilibrer l'avancement des horloges locales (*partie 2*).

Au niveau du contrôle global, nous proposons un nouvel algorithme LT qui détermine un minorant de la date possible des rollbacks pour chaque PL: la date locale sûre ou LT. Cet algorithme corrige les défauts majeurs des algorithmes basés sur la date globale virtuelle GVT existants. Contrairement à ceux-ci, il détermine la valeur de LT des PLs sans stopper leurs activités et sans avoir à examiner simultanément l'ensemble des PLs du système. Par conséquent, nous avons éliminé l'organe central de calcul des algorithmes de GVT ainsi que le réseau supplémentaire de communication nécessaire pour déterminer GVT, tout en repectant au mieux les paradigmes de de modèle réparti. Nous obtenons ainsi un raffinement des algorithmes de GVT permettant dans tous les cas de réduire la taille de mémoire requise (partie 3).

1.7 Plan de la thèse

La thèse est organisée en quatre parties (cf. figure 1.1):

La *première partie* présente une synthèse des méthodes de la simulation répartie asynchrone par événements discrets. Elle étudie la modélisation de l'application simulée en terme de processus communiquant par messages et la définition de schémas d'exécution répartis. Deux stratégies de synchronisation sont étudiées: l'une sûre et l'autre optimiste (risquée). Le protocole de synchronisation optimiste est basé sur la notion de temps virtuel [Jef85] et comporte deux mécanismes de contrôle: le contrôle local et le contrôle global.

La *deuxième partie* présente notre nouveau modèle de contrôle adaptatif d'agent de la simulation répartie: ses concepts et sa réalisation. Cette présentation est précédée de l'étude des différents travaux existants pour maîtriser le mécanisme de synchronisation optimiste: la distorsion temporelle ("Time Warp").

La *troisième partie* présente un nouvel algorithme de contrôle global, *l'algorithme LT* puis la preuve formelle de son exactitude. Cette présentation est précédée d'une étude des différents travaux sur le mécanisme de contrôle global basé sur la notion de l'horloge virtuelle globale GVT ("Global Virtual Time").

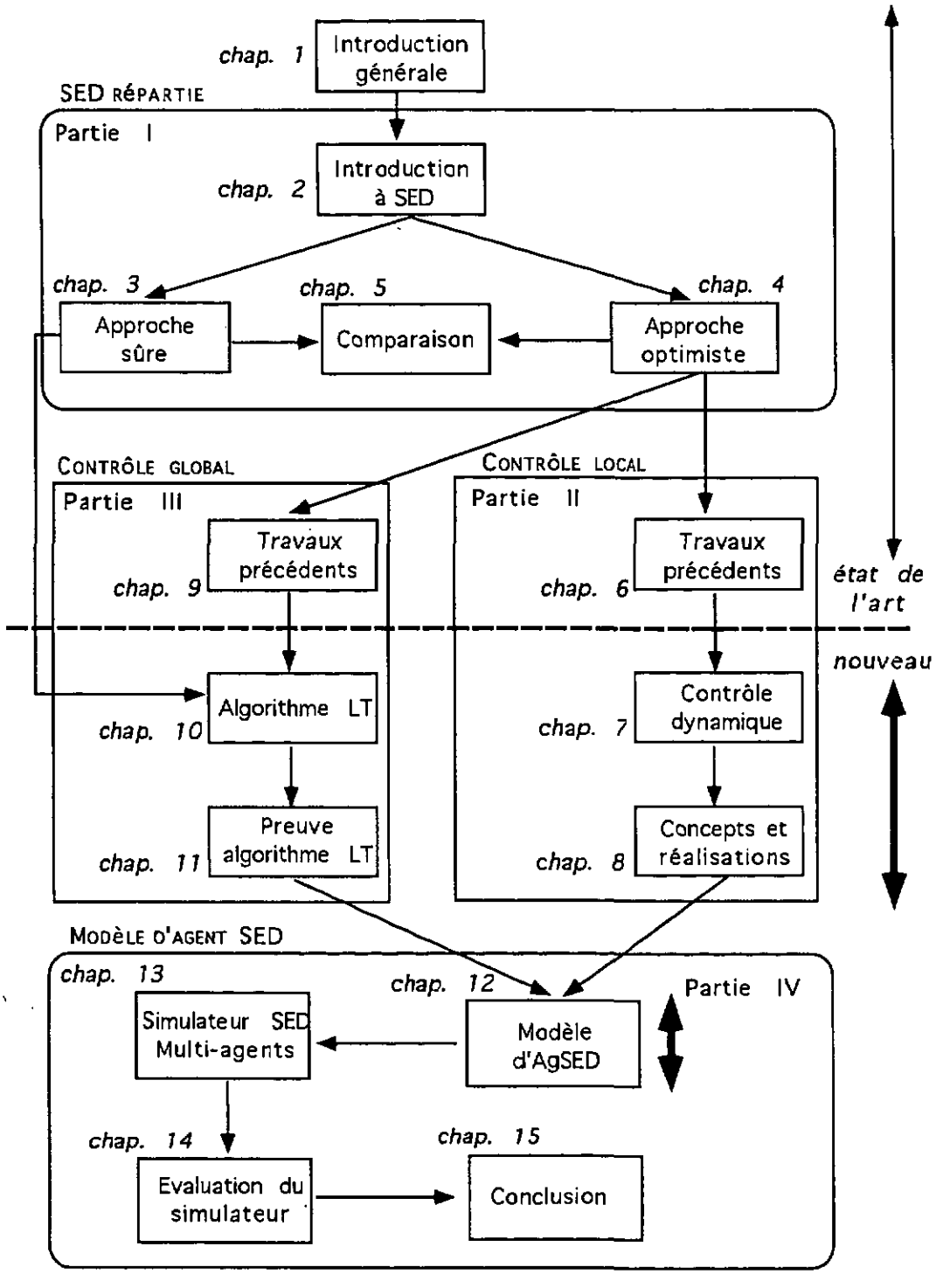


Figure 1.1: Structure hiérarchique des chapitres

La *quatrième partie* est à la frontière de deux domaines en informatique: la technique de modélisation et l'intelligence artificielle distribuée. Plus précisément, elle relie la simulation par événements discrets et les Systèmes Multi-Agents (SMA). Elle englobe les améliorations exposées dans les parties précédentes.

Organisation de la première partie

Cette partie comporte quatre chapitres.

Le premier chapitre (chapitre 2) présente les propriétés que doit remplir un modèle exécutable. Elle introduit les mécanismes de base d'un programme séquentiel pour cerner les problèmes de synchronisation qu'on rencontre dans une simulation répartie. Dans la simulation répartie, le modèle est décomposé en plusieurs parties qui interagissent. Chaque partie est simulée sur un processus et les processus communiquent par échanges de messages. Les processus sont affectés aux processeurs de l'architecture parallèle ou répartie pour l'exécution. La mise en œuvre de la simulation suit le principe de CSP ("Communicating Sequential Processes") [Hoa78] où les processus séquentiels communiquent par des primitives d'envoi et de réception de messages. Cette répartition de la simulation nécessite un protocole de synchronisation des processus pour que l'ordre chronologique des événements dans le système (ou contrainte de causalité) soit respecté. Les différentes stratégies de synchronisation se résument à deux: l'une **sûre** ("conservative approach") et l'autre **optimiste** ou risquée ("optimistic approach").

Le deuxième chapitre (chapitre 3) étudie le mécanisme de synchronisation sûre où un processus ne traite un événement que lorsqu'il est assuré que cette opération ne viole pas la contrainte de causalité.

Le troisième chapitre (chapitre 4) étudie le mécanisme de synchronisation risquée où un processus peut violer la contrainte de causalité. Mais une fois qu'elle n'est plus respectée, le processus doit pouvoir rétablir son dernier état respectant la contrainte de causalité et reprendre la simulation.

Le quatrième chapitre (chapitre 5) présente la comparaison de deux mécanismes de synchronisation de la simulation répartie.

Organisation de la deuxième partie

Notre étude se base sur la simulation optimiste qui ne pose aucune contrainte sur les processus. Nous y introduisons la notion de distorsion temporelle qui comporte deux mécanismes de contrôle: le contrôle local et le contrôle global.

La deuxième partie est entièrement consacrée au contrôle local dans la distorsion temporelle. Elle est structurée en trois chapitres:

Le premier chapitre (chapitre 6) présente les travaux antérieurs et les fondements de l'amélioration du mécanisme de contrôle local pour rendre le simulateur efficace. Notre axe de recherche pour améliorer le mécanisme de contrôle local découle de la synthèse des travaux précédents.

Le deuxième chapitre (chapitre 7) présente notre nouveau modèle de contrôle adaptatif

ou dynamique. Ensuite, il prouve à partir de la théorie de contrôle adaptatif comment ce modèle stabilise le simulateur.

Le troisième chapitre (chapitre 8) présente la conception et la réalisation du modèle de contrôle adaptatif dans la distorsion temporelle. Il intègre dans la distorsion temporelle normale la mise en œuvre du contrôle adaptif qui corrige au cours de l'exécution les défauts des mécanismes statiques existants.

Organisation de la troisième partie

La troisième partie est entièrement consacrée au contrôle global dans la distorsion temporelle qui permet de valider l'ensemble de la simulation jusqu'à la date GVT. Cette partie est structurée en trois chapitres.

Le premier chapitre (chapitre 9) présente l'état de l'art des algorithmes de détermination de GVT et les défauts des algorithmes existants sont relevés.

Le deuxième chapitre (chapitre 10) présente une approche inédite de détermination de GVT au sein de chaque PL. Nous proposons un algorithme qui détermine localement l'avancement de la simulation sans consulter l'ensemble des PLs dans le simulateur mais seulement une partie. Chaque processus dispose d'une variable locale interne LT ("Safe Local Time") équivalente à GVT.

Le troisième chapitre (chapitre 11) présente la preuve formelle de l'*algorithmeLT* proposé au chapitre précédent.

Organisation de la quatrième partie

La quatrième partie est structurée en trois chapitres et la conclusion générale.

Le premier chapitre (chapitre 12) présente un modèle d'agent de la simulation par évènements discrets répartie: AgSED. Ce modèle engloble les concepts exposés dans les parties précédentes, notamment le contrôle adaptatif (cf. chapitre 7 et 8) et la date locale sûre (cf. chapitre 10 et 11) dans la mise en œuvre de la distorsion temporelle.

Le deuxième chapitre (chapitre 13), après une brève introduction à la simulation multi-agents, présente la réalisation du modèle d'agent AgSED dans un environnement réparti simulé en smalltalk.

Le troisième chapitre (chapitre 14) expose les applications de réseaux de files d'attente sur lesquelles nous évaluons la performance des concepts développés dans l'environnement mult-agents simulé.

Le dernier chapitre (chapitre 15) présente notre contribution et les axes de recherche à approfondir.

Partie I

Simulation répartie

Chapitre 2

Introduction à la simulation par événements discrets (SED)

2.1 Introduction

L'étude de la simulation d'un système discret comporte quatre étapes [Mis86]:

1. l'observation du système réel et la compréhension de ses caractéristiques;
2. la construction du modèle approprié au système dans lequel on retient uniquement les caractéristiques pertinentes par rapport aux résultats recherchés;
3. la construction du modèle exécutable à partir du modèle et l'exécution sur une machine; l'élaboration d'un schéma d'exécution pour faire évoluer le modèle: *le simulateur*;
4. l'analyse des résultats obtenus lors de l'étape précédente, la compréhension et la prédiction du comportement du système réel.

L'activité de la simulation consiste en fait en un va-et-vient entre ces quatre étapes: l'analyse des résultats peut conduire à effectuer de nouvelles expériences ou à remettre en cause le modèle parce qu'il n'est pas assez précis ou parce qu'il ne capte pas assez bien les caractéristiques intéressantes du système réel.

L'*objet de notre étude se rapporte à la phase 3 de construction du programme*. Nous nous intéressons à la manière de passer du modèle ou du système physique à la réalisation du programme exécutable de simulation.

Dans la section 2, les principes du système physique qui nous intéressent et les propriétés de leur modèle exécutable sont précisés. La section 3 aborde la problématique de la description d'un modèle d'exécution pour la simulation par événements discrets: le simulateur. La section 4 décrit les éléments essentiels de la réalisation d'un simulateur séquentiel pour montrer la nécessité d'un modèle d'exécution parallèle. La section 5 étudie le paradigme des processus logiques (PLs). La section 6 présente les différentes approches possibles aussi bien algorithmiques et logicielles que matérielles pour accélérer

l'exécution parallèle d'un simulateur dans un environnement réparti. La section 7 présente les caractéristiques de la simulation répartie et précise le genre de problèmes qui nous intéressent dans la réalisation d'un simulateur réparti. La section 8 introduit les notions de relation de causalité et de contrainte de causalité desquelles découlent les événements pouvant s'exécuter en parallèle ou de façon séquentielle dans un simlateur réparti.

2.2 Caractéristiques du système et modèle simulé

2.2.1 Principes du système physique

Les système physiques qui nous intéressent obéissent à deux principes :

Principe 1 (Principe de causalité)
Le futur ne peut influencer le passé. Plus précisément, l'état du système à l'instant t est indépendant de tout ce qui peut se produire à une date t' ultérieure à t.

Principe 2 (Principe de déterminisme)
Le futur du système peut être déterminé à partir de son état présent et de son passé. En d'autres termes, à tout instant t, il existe une valeur positive ε telle que le comportement futur du système puisse être calculé jusqu'à $t + \varepsilon$.

Ces deux principes, qui gouvernent l'évolution des systèmes physiques, portent sur le temps (*causalité*) et sur la loi d'évolution du système (*déterminisme*). Toute modélisation doit respecter le principe de déterminisme et tout simulateur doit garantir le principe de causalité. Le système peut être décomposé en entités, chacune dotée d'un état et d'un comportement propres. Ces entités interagissent pour produire le comportement global du système.

Nous nous intéressons essentiellement au type de modélisation ne comprenant que des *entités actives*. Comme le concept de processus est un outil efficace pour en représenter le comportement, on parle alors de *processus physique* notés PP. Les interactions entre ces processus peuvent être représentées de diverses manières mais elles impliquent souvent un échange d'informations. Nous suivons ici l'approche de Chandy et Misra [CM79b], [Mis86] qui proposent de modéliser ces interactions par des échanges de messages, ce qui assure, d'une part, une bonne capacité d'expression pour le modèle et, d'autre part, une transposition aisée dans le contexte réparti.

Un modèle ou système physique est représenté par un graphe dont les sommets sont les processus physiques. Deux processus sont reliés s'ils décrivent des entités interagissant dans la réalité. La communication entre les processus est de durée nulle dans le modèle simulé.

2.2.2 Propriétés des modèles exécutables

Les principes auxquels obéissent les systèmes physiques étudiés (causalité et déterminisme) ont des conséquences sur leur modélisation, c'est-à-dire sur la forme du réseau de processus logiques (PLs) constituant le modèle exécutable et sur ses règles d'évolution. Dans le

modèle exécutable chaque processus physique est représenté par un processus logique où l'échange d'information entre les processus physiques est représenté par des messages d'é vénement à travers un canal orienté. Le modèle exécutable satisfait les conditions suivantes [Mis86]:

Condition 1 (Condition de réalisabilité ("realizability condition"))
Un message envoyé par un PL à l'instant (simulé) t est une fonction de son état initial et des messages qu'il a reçus jusqu'à t (inclus). Un message reçu après t ne peut avoir d'effet sur le comportement du PL jusqu'à t.

Cette condition du modèle simulé est le pendant du principe de causalité dans le système réel.

Condition 2 (Condition de prédicabilité ("predictablity condition"))
S'il existe un circuit sur le réseau des PLs, alors, à tout instant t, il existe un PL de ce circuit dont les messages envoyés peuvent être calculés jusqu'à $t + \varepsilon$ (avec $\varepsilon > O$) en fonction de son état initial et des messages qu'il a reçus jusqu'à t inclus.

En d'autres termes, sur un circuit, il existe à tout instant au moins un processus ayant un temps de service non nul, tel que la réception d'un message à l'instant t provoquera l'émission d'un message à t' avec $t' > t$. Cette condition du modèle est le pendant du principe de déterminisme imposé au système réel.

2.3 Programme de simulation

Une fois le modèle du système réel conçu, il faut le décrire à l'aide d'un programme. Le problème principal posé par cette description est la manipulation du temps simulé. Deux types d'expression sont possibles:

- l'expression par **événements**; chaque événement est décrit par une condition (la condition d'exécution de l'événement) et par les instructions qui représentent le changement d'état correspondant; le programme est alors composé de la description des structures de données représentant l'état de chaque composant du système et de la description de tous les événements possibles; les actions associées à l'exécution d'un événement sont toujours de durée nulle dans le temps simulé;
- l'expression par **processus**; les processus du modèle sont alors représentés dans le programme de simulation par des processus qui exécutent des actions matérialisant les changements d'état, l'évolution du temps et les interactions entre les processus; dans ce cas, certaines de ces actions ont une durée non nulle dans le temps simulé.

Dans le cas d'une description sous forme de processus, les interactions entre processus du programme de simulation peuvent être de différentes natures:

- l'*interaction par ressource*; une ressource est un objet passif utilisé par les processus du programme de simulation. Ce type de description est adapté à la modélisation *client-serveur*;

- l'*interaction directe*; les processus du programme de simulation peuvent s'activer ou se désactiver directement par un ensemble de primitives mises à disposition [PBH+88], [LK91]. Simula [DMN70] est le représentant bien connu de cette classe;
- l'*interaction par message*; les processus du programme de simulation émettent et reçoivent des messages [Mis86], [J+87], [BCM87].

Précisons la sémantique des émissions et réceptions de messages: les messages reçus par un processus logique sont stockés dans une file et retirés au fur et à mesure des demandes. Ainsi, un message émis par PL_i vers PL_j à une date t (dans le temps simulé) est disponible à t dans la file de réception de PL_j. Son traitement effectif par PL_j à sa demande sera effectué à une date ultérieure t' avec $t' > t$.

Un processus logique peut donc exécuter quatre types d'actions:

- la mise à jour des variables décrivant son état. Cette action a une durée nulle dans le temps simulé.
- la mise en œuvre de l'écoulement du temps simulé (correspondant à la durée des actions dans la réalité). Cette synchronisation par rapport au temps simulé peut se traduire par la construction *attendre* ($Tsim = t$), où $Tsim$ représente la valeur du temps simulé.
- l'émission d'un message, de durée nulle dans le temps simulé.
- la synchronisation avec d'autres processus, représentée par l'attente d'un message. Cette action a une durée éventuellement non nulle et non connue à l'avance – a priori – dans le temps simulé. Elle peut se traduire par une construction de la forme *attendre (message m)*.

2.3.1 Simulateur correct

Le résultat de la modélisation du système physique est un réseau de processus physiques PPs communiquant par messages [Mis86]. Ces processus évoluent dans le temps; le temps de communication des messages (interactions) est de durée nulle. Ceci est vrai quels que soient les outils utilisés pour décrire ces PPs sous forme de programme informatique. Réaliser un simulateur consiste à l'exécuter de telle façon que son temps de référence soit le temps simulé (et non le temps de la machine supportant l'exécution), ce qui inclut, d'une part, l'avancement du temps et, d'autre part, le respect des relations de causalité. Nous appellerons processus logique, noté PL, la suite d'actions exécutées sur le simulateur pour représenter l'évolution d'un processus physique PP du modèle . La ***simulation est dite correcte*** s'il lui est possible de produire, grâce aux PLs, les séquences exactes de messages communiqués entre PPs, et les dates associées [Mis86].

2.3.2 Différents types de messages

Le simulateur réparti propose la répartition des PLs sur différents processeurs, puis exige que les PLs puissent communiquer entre eux. Alors, la mise en place des PLs sur différents

processeurs nécessite la création et la gestion d'une liste d'événements et d'une horloge locale dans chaque processeur. La recherche sur la simulation parallèle se focalise avant tout sur la coordination de l'avancement des horloges locales pour garantir un même résultat qu'une exécution séquentielle.

Chaque *PL* peut envoyer à un autre *PL* deux types d'information: soit un *message d'événement* soit un *message de contrôle*. Un ***message d'événement*** transporte une information qui se rapporte au changement dans le système physique simulé. Un ***message de contrôle*** ou de non-événement correspond au message simulé pour contrôler le simulateur. Ces derniers ne correspondent à aucune information dans le modèle simulé.

2.3.3 Mécanisme d'avancement du temps simulé

La réalisation d'un simulateur réparti implique la définition d'un schéma d'exécution réparti qui assure la progression du temps simulé (*la propriété de vivacité ou de vitalité*) en respectant les relations de causalité du système simulé (*la propriété de sûreté ou d'intégrité*). Le problème fondamental dans la conception d'un simulateur est donc la gestion du temps simulé. Il s'agit d'en trouver une représentation et de construire le schéma d'exécution qui en assure une gestion correcte. Deux grandes classes de solutions ont été proposées pour avancer le temps simulé *Tsim*:

1. La *simulation conduite par le temps*:

   Le simulateur avance la variable *Tsim* (*horloge de la simulation*) par pas de δt avec $\delta t > 0$. A chaque incrémentation, il examine les *PL*s et exécute les actions d'événement impliquées par leur état et la valeur courante de l'horloge *Tsim*.

   Lorsqu'un δt est assez grand, on perd l'ordre des événements survenus en une unité de temps simulé. Par contre, pour un δt assez petit, on effectue fréquemment des traitements inutiles, ce qui augmente le temps d'exécution. Le choix du pas est un compromis entre la précision souhaitée et l'efficacité désirée du simulateur.

   La simulation conduite par le temps peut être synchrone ou asynchrone. Dans le premier cas, tous les processeurs doivent achever de simuler les événements correspondants au δt courant avant de passer au suivant. Ceci se fait soit par un processeur central contrôlant l'activité de tous, soit par un mécanisme de synchronisation réparti consistant en des échanges de messages pour communiquer la fin des calculs pour le δt courant. Dans le cas d'une simulation asynchrone, chaque processeur a sa propre horloge et peut commencer la simulation de δt suivant tout de suite aprés avoir fini la simulation du δt courant.

2. La *simulation conduite par les événements*

   Le simulateur évalue le premier instant *tsuiv* dans le futur où se produira une modification de l'état du système (événement), affecte à *Tsim* la valeur *tsuiv* et effectue les actions correspondant à l'événement sélectionné. Ces actions peuvent produire de nouveaux événements pour le futur ou en supprimer.

   La simulation conduite par événements peut être synchrone ou asynchrone. Dans le premier cas, une horloge globale est mise à jour à la plus petite date d'exécution de

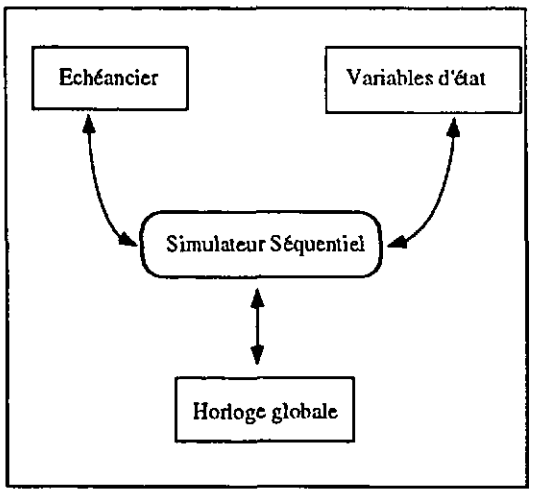


Figure 2.1: Simulateur séquentiel

tous les événements. Le schéma synchrone est réalisé par un processeur central ou par une approche répartie à l'aide des messages échangés entre les PLs pour fixer l'instant de la simulation. Dans le second cas, chaque PL possède son horloge locale. La nature synchrone du modèle réel est maîtrisée en représentant le temps simulé dans chaque message (m, t) transmis d'un processus logique à un autre. Ainsi, les PLs s'exécutent parallèlement aussi longtemps que leurs contreparties en système physique s'exécutent indépendamment.

Les PLs avancent localement dans le temps selon deux protocoles de synchronisation qui seront développés ultérieurement: l'un sûr et l'autre optimiste.

La simulation conduite par le temps est moins efficace que la simulation conduite par événements, d'autant plus qu'il peut arriver qu'on avance de plusieurs pas (incréments de temps) sans que rien ne se passe. La simulation par événements est plus souple et beaucoup plus efficace, toute action du simulateur fait progresser la simulation.

2.4 Simulation séquentielle

Un simulateur séquentiel classique utilise trois types de structure de données (figure 2.1).

- les variables d'états qui décrivent l'état du système;
- une liste d'événements ou échéancier qui contient l'ensemble des événements dont les dates d'exécution ont été planifiées mais n'ont pas encore pris effet;
- une horloge globale qui indique la manière dont la simulation avance.

Un changement d'état ou un événement se produit à un instant t dans le système réel simulé. Cet instant t auquel le simulateur doit traiter les actions liées à l'événement, vu le rôle important qu'il va jouer par la suite, sera appelé ***estampille de l'événement*** ou du message d'événement.

Les actions à exécuter dans un simulateur séquentiel sont planifiées à l'aide des événements dans l'échéancier. Elles sont triées dans l'ordre croissant de leurs estampilles.

```
        EventQueue désigne l'échéancier;
        GlobalClock désigne l'horloge globale de la simulation
begin
        Initialisation:
            introduit le premier événement dans EventQueue
        WHILE NOT empty(EventQueue) DO
            currentEvent := GetFrom(EventQueue);
            UpdateGlobalClock(currentEvent);
            Simulate(currentEvent);
            FOR each generated event DO
                InsertIn(event,EventQueue);
            END FOR
        END WHILE
end.
```

Figure 2.2: *Structure algorithmique d'un simulateur séquentiel*

A chaque cycle, la *boucle principale* (cf. 2.2) du simulateur extrait le premier événement (ayant la plus petite estampille) de l'échéancier ou l'événement courant et le traite de la manière suivante:

- donner à l'horloge globale l'estampille de l'événement courant;
- simuler les actions rattachées à l'événement courant. Le traitement de cet événement implique l'exécution d'un code de programme qui simule le changement survenu dans le système simulé. Il planifie le cas échéant un ou plusieurs événements dans le futur simulé en ajoutant ou supprimant au besoin des évènements de l'échéancier. Ainsi, l'événement courant définit les relations de causalité dans le simulateur.

Les simulateurs séquentiels évolués ou *orientés processus* [ED96], [LK91] [PBH+88] sont construits selon le mécanisme de base ci-dessus où l'échéancier manipule des processus au lieu d'événements.

Dans ce paradigme d'exécution, il est crucial de toujours sélectionner dans l'échéancier l'événement e_{min} ayant la plus petite estampille et de le traiter. Sans quoi, en exécutant d'abord un événement e de date ultérieure, on pourrait modifier les variables d'état utilisées par e_{min}. Ceci reviendrait à simuler un système dans lequel le futur peut affecter le présent, ce qui est inacceptable d'après le principe de causalité. Les erreurs de cette nature sont appelées ***erreurs de causalité***.

Caractéristiques d'une exécution parallèle

La caractéristique du parallélisme est de pouvoir traiter les événements sur différents processeurs sans horloge globale pour synchroniser les actions du modèle. Une transposition

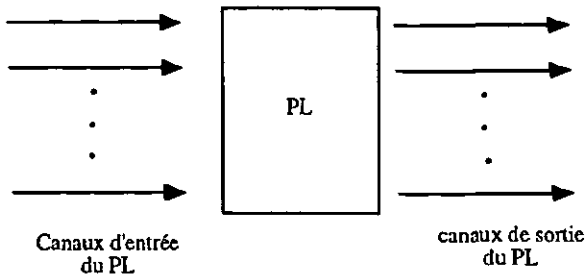


Figure 2.3: Structure d'un PL

directe du paradigme séquentiel pose déjà de sérieuses difficultés sur un ordinateur multiprocesseurs. Considérons l'exécution concurrente de deux événements e_1 et e_2 dont les estampilles sont respectivement t_1 et t_2, avec $t_1 < t_2$. Si e_1 écrit dans une variable d'état qui est lue plus tard par e_2, alors e_1 doit s'exécuter avant e_2 pour éviter qu'une erreur de causalité ne survienne. En d'autres termes, certaines *suites de contraintes* doivent être maintenues pour que les calculs simulés soient corrects. La plupart des stratégies de modélisation répartie existantes évitent le scénario ci-dessus en utilisant une méthodologie de processus logiques appelée le *paradigme de processus logiques* qui interdit strictement aux processus physiques de partager des variables d'état. Certaines modélisations font exception en autorisant le partage des variables d'état [Fuj89b], [Jon86], [JCRB89].

2.5 Paradigme de processus logiques

Le modèle exécutable, ou système physique, est ici constitué d'un ensemble fini de processus physiques ou PPs qui interagissent par des canaux unidirectionnels à différents instants discrets de la période de simulation. On définit ainsi un réseau dont les sommets sont les processus physiques et les arcs, des lignes de communication orientées par lesquelles les processus physiques interagissent.

Le simulateur est réalisé à partir du système logique. Celui-ci est construit à partir du système physique de la manière suivante: à chaque processus physique PP est associé un processus logique PL. Une voie de communication orientée du processus logique PL_i vers le processus logique PL_j est définie chaque fois qu'un processus physique PPi peut envoyer un message à un autre PP_j. Les interactions entre les PLs sont modélisées à l'aide de messages d'événement qui contiennent la date où l'événement se produit (*estampille*). Ainsi un message m envoyé dans le système physique d'un processus physique PP_i à un autre PP_j à l'instant t est représenté par le message (m, t) envoyé du PL_i à PL_j par le canal orienté qui les relie.

Chaque processus logique contient l'état de son processus physique modélisé et une horloge locale indiquant jusqu'où le processus logique a progressé au cours de la période de simulation.

Définition 1 (canal de sortie et canal d'entrée)
Parmi les canaux d'un processus logique PL (cf. figure 2.3), on distingue:

- *ceux qui aboutissent au PL, appelés canaux d'entrée de PL;*

- *ceux qui partent de PL, appelés canaux de sortie de PL.*

Remarque 1 (*PL* source et *PL* puits)
Un processus logique qui n'a pas de canaux d'entrée est appelé *PL source* : il ne fait qu'envoyer de messages sans en recevoir. Par contre un *PL* qui n'a pas de canal de sortie ne fait que recevoir des messages: il est appelé *PL puits ou sortie.*

2.5.1 Problème de terminaison instantanée

Dans un simulateur séquentiel, on peut affirmer que le simulateur a complètement achevé son traitement à un instant t donné grâce à l'horloge globale du simulateur. Par contre, dans un simulateur réparti, on ne maîtrise pas la vision globale de l'évolution du système puisque chaque *PL* simule localement son *PP* correspondant. On définit la valeur de l'horloge du système par la plus petite valeur entre (i) le minimum des des horloges locales et (ii) la plus petite estampille des messages en transit en attente de traitement.

2.5.2 Encapsulation de *PP*s par des *PL*s

La différence fondamentale entre un simulateur réparti et les simulateurs séquentiels est le manque de contrôle global du système. Le simulateur est correct quand chaque *PL* simule correctement son *PP*. Par conséquent, pour réaliser un simulateur complexe correct, il suffit de le partitionner en de simples simulateurs corrects [Mis86].

2.5.3 Analyse du paradigme de *PL*s

Supposons que le système logique comporte n processus logiques, $PL_1, \ldots, PL_n$ où t_i est l'horloge locale de chaque PL_i; chaque message d'événement contient une estampille (la date où l'événement doit se produire).

Le paradigme de processus logiques exige du programmeur d'application une discipline de partition des variables d'état du simulateur en un ensemble d'états disjoints pour assurer que chaque *PL* n'accède qu'à ses propres variables. Est-ce vraiment une manière naturelle de programmer le simulateur? Il est vrai que l'on pourrait réaliser les *variables partagées* par envoi de messages en les dupliquant dans les *PL*s concernés. Mais cette approche ne nuit-elle pas à la performance du système?

L'exclusion des variables partagées, selon l'application, peut être un avantage ou un inconvénient. Par exemple, elle est bien adaptée à un système de file d'attente car le comportement de chaque serveur est indépendant de l'état des autres. Par contre, elle est mal adaptée aux problèmes tels que la simulation d'un champ de bataille où les unités de combat (PL) se déplacent sur un terrain commun et interagissent occasionnellement avec les autres. Une approche naturelle pour modéliser ce système est de partitionner le champ de bataille en une grille de dimension 2 indiquant la position de chaque unité de combat. Les unités de combat doivent scruter l'environnement pour attaquer, se retirer ou changer de position sur la grille. Les informations de l'environnement indiquent les unités de combats qui résident dans chaque secteur de la grille. Cette approche, en l'absence

de variable partagée (qu'est ici la grille), conduit souvent à une faible performance du modèle à cause du coût élevé de communication.

Remarque 2
Lorsque les informations partagées sont dupliquées entre les *PLs*, il est alors nécessaire d'assurer la cohérence des variables partagées (l'intégrité des variables) lorsqu'une variable est modifiée dans un *PL*. Cette approche peut être efficace pour accomplir une bonne performance. Cependant, elle complique considérablement le code de l'application, sa compréhension et sa mise à jour.

2.6 Parallélisation de la simulation

Compte tenu de la complexité des systèmes étudiés, la simulation exige un temps de calcul très élevé, alors que l'obtention rapide de ses résultats a toujours été nécessaire. On cherche donc tous les moyens possibles [MF94], aussi bien algorithmiques et logiciels que matériels, pour accélérer son exécution.

2.6.1 Architecture cible des machines

Grâce à l'évolution technologique de ces dernières années, différents types d'architectures matérielles, chacune plus ou moins adaptée à une famille d'applications et de problèmes, sont actuellement disponibles sur le marché [Fis95], [TW91]. De plus il est possible de concevoir et de réaliser une architecture spécifique pour la résolution d'un problème particulier. L'utilisation de ces architectures pour la simulation a donc donné naissance à plusieurs axes de recherche dont

- les machines vectorielles et de type SIMD (Single Instruction Multiple Data) pour la simulation des systèmes continus et des systèmes à événements discrets équationnels, et
- les machines de types MIMD (Multiple Instruction Multiple Data) pour la simulation par événements discrets.

Misra [Mis86] a prouvé que la simulation à événements discrets séquentielle classique, utilisant un échéancier comme nous l'avons vu plus haut, est correcte, c'est-à-dire qu'elle représente bien le fonctionnement du système réel. Il montre également que la structure même de l'échéancier reflète la nature séquentielle du problème. Ainsi, la simulation par événements discrets d'un système a été traditionnellement réalisée de manière séquentielle. Ceci s'avère de plus en plus pénalisant pour les performances globales de la simulation car plus le système est complexe, plus la quantité d'événements à traiter augmente.

Etant donné la séquentialité imposée par la méthode de l'échéancier dans une simulation séquentielle, on s'intéresse aux méthodes de parallélisation qui permettent de tirer un réel avantage du parallélisme de la machine-cible. Les différentes méthodes développées pour paralléliser la simulation par événements discrets peuvent être classées en trois catégories: *l'accélération matérielle*, *la parallélisation temporelle* et *la parallélisation spatiale*.

2.6.2 Accélération matérielle

L'accélération matérielle consiste à concevoir une carte spéciale pour un type particulier d'application. Les performances globales de la simulation utilisant une décomposition spatiale peuvent souvent être fortement dégradées par les protocoles de synchronisation des processus. Différentes études ont donc été menées pour mettre au point des composants ou des matériels spécifiques à la simulation. Wu [Wu90] résume les différentes approches possibles: processeur avec un jeu d'instructions spécifique à la simulation (par exemple la manipulation de l'échéancier), architecture spécifique permettant la technique de répartition des fonctions [RW89a], machine multiprocesseur spécifique à chacun des protocoles cités plus haut.

Concepcion [Con89] propose une architecture à bus hiérarchique: le simulateur est spécifié de manière hiérarchique et appliqué directement sur l'architecture. Blesh et Arpasi [BA85] décrivent également une architecture à base de bus, munie d'un système d'exploitation et d'un langage spécifiques, l'ensemble étant destiné à la simulation en temps réel dans le domaine de l'aéronautique.

Lubachevsky [Lub88] et Reynolds [Rey91] ont proposé l'adjonction d'un nouveau réseau de contrôle sur une architecture multiprocesseur existante. Ce réseau de contrôle, dont la topologie est un arbre binaire, permet une accélération importante des opérations globales (collecte, diffusion). Dans la même idée, Filloque [Fil92] décrit des algorithmes d'opération et de synchronisation globales pour les différents types de simulation (synchrone et asynchrone) utilisant une machine MIMD à mémoire distribuée munie d'un réseau logique programmable configuré en anneau, et permettant des opérations globales rapides.

Fujimoto et al. [FTG88b], [FTG92] proposent un composant spécifique, *roll-back chip*, destiné à la gestion des états dans *time warp*. Ce composant a été intégré par la suite dans un système industriel à base de bus [BRF90] et constitue le noyau de l'architecture de VTM (virtual time machine) [GF91] destinée aux applications utilisant le **temps virtuel** [Jef85]. Cette architecture permet un accès direct à la mémoire selon l'adresse et la date virtuelle des données. L'accélération matérielle est très coûteuse et s'adapte mal à un environnement de simulation par événements discrets.

2.6.3 Décomposition temporelle

D'autres décompositions, dites *temporelles* ont été développées et ont conduit au *parallélisme temporel*. L'idée est de simuler en parallèle le comportement d'un même objet dans différents intervalles de temps. Cette méthode est développée par Chandy et Sherman [CS89a]. Le modèle simulé est représenté dans un graphe d'espace à deux dimensions *(x,t)=(espace, temps)* où l'axe *espace* comporte l'ensemble des variables d'état utilisées dans la simulation et second axe *temps* le temps auxquels les variables sont simulées. Ils observent que l'horizon d'une simulation est un point fixe, c'est-à-dire qu'à la fin, même si la simulation se poursuit, l'état observé du modèle ne change plus. Le graphe d'espace *(espace, temps)* peut alors être divisé en régions qui seront chacune simulée par un processus. Les processus voisins s'échangent des messages. Le traitement dans chacun d'eux itère sur l'état de ses voisins et des *suppositions* sur ces états pour converger

vers le point fixe. Cette idée de partition et de simulation par supposition, suivie d'une *correction* ("fix-up") éventuelle a également été développée par la suite dans [AD91] et [LL91b] pour certains types de modèles. La simulation de systèmes particuliers, dont l'évolution s'exprime sous forme d'équations de récurrence dans le temps, a aussi été étudiée récemment dans [GLM91], [GGN92], [BC93]. La décomposition de la simulation permet dans ce cas d'obtenir des accélérations importantes sur les machines massivement parallèles. Toutefois, cette approche peut conduire à des difficultés logiques de division et de partitionnement sur les processeurs.

2.6.4 Décomposition spatiale

La décomposition du modèle à simuler se fait selon sa structure topologique. Les différentes parties du modèle sont réparties sur l'architecture-cible. Les stratégies envisageables pour paralléliser une simulation [RW89a] sont:

La parallélisation par compilation:

un compilateur essaie de détecter les parties de code pouvant être exécutées en parallèle. Cette solution qui ignore la structure du problème ne permet d'exploiter qu'une faible partie du parallélisme potentiel [PM89].

La répartition des expériences :

on garde un programme séquentiel pour simuler le modèle mais on utilise plusieurs processeurs en faisant plusieurs expériences en même temps (varier les paramètres du modèle ou la graine des nombres pseudo-aléatoires utilisés) [BDO85]. Cette solution est efficace car les processeurs n'interagissent que pour faire des études statistiques (moyenne des résulats partiels). Mais elle manque de souplesse et nécessite la mise à disposition sur chaque processeur d'une taille de mémoire suffisante pour décrire tout le modèle. Cette approche est très efficace lorsqu'on s'intéresse au comportement transitoire du modèle. Par contre, dans le cas où l'on s'intéresse uniquement à son état stationnaire et que le modèle simulé met longtemps à atteindre cet état, la technique devient totalement inefficace dans la minimisation des erreurs d'estimation statistique car elle est inutilement lente dans la période transitoire. Les aspects statistiques de ce type de simulation parallèle sont étudiés en détail dans [Hei88] et [GH91].

La répartition des fonctions du simulateur:

on affecte à chaque processeur la réalisation d'une classe de fonctions du simulateur (génération des variables aléatoires, traitement des événements, prise des mesures et traitement statistique associé, etc.) [Wya85], [SDC88] et [CG88]. Cette méthode ne permet d'utiliser efficacement qu'un nombre réduit de processeurs.

La répartition des fonctions événements:

on maintient un échéancier global et dès qu'un processeur devient libre, on lui fait traiter le premier événement de cet échéancier. Cette stratégie est particulièrement bien adaptée aux multiprocesseurs à mémoire partagée [CS88], [Jon86]. Un mécanisme de gestion de cohérence des données est nécessaire car le traitement des événements courants

dans les différents processeurs peut affecter le traitement d'un événement dans l'échéancier.

La répartition des composants du modèle:

le modèle est décomposé en processus physiques qui interagissent [Fer95]. Chaque processus physique est simulé par un processus logique *PL*. Les *PL*s sont placés sur les processeurs selon la configuration matérielle disponible. Un processeur peut être chargé de l'exécution de plusieurs processus logiques. Les interactions entre ceux-ci se font par échange de messages sur le réseau d'interconnexion de l'architecture cible. Cette technique, appelée **simulation répartie**, est la plus intéressante car elle peut exploiter au maximum le parallélisme inhérent au modèle simulé. Compte tenu de l'indépendance de chaque processeur, un protocole de synchronisation est nécessaire pour garantir la causalité dans la simulation. Deux types de protocoles sont possibles : protocole synchrone et protocole asynchrone.

Le principe de base des protocoles synchrones est le suivant: *Avant le traitement d'un événement local, un processus doit se synchroniser avec tous les autres. Les événements, et donc les processus concernés, sont choisis lors de cette phase de synchronisation: il représente les événements portant la date minimale. Les événements ayant la même date sont donc simulés en parallèle s'ils sont situés sur des sites différents.* Différentes variantes de ce protocole ont été développées [SBW88], [Lub88], [CS89b], [Aya89] et [Ste91]. La plupart d'entre elles sont basées sur la technique de la *fenêtre temporelle* pour affaiblir la synchronisation imposée aux processus. Une fenêtre de taille fixe ou variable est définie à chaque étape, et chaque processus avance parallèlement aux autres dans cette fenêtre.

Les deux protocoles asynchrones, sûr [CM79a] et optimiste [Jef85], feront l'objet des deux prochains chapitres.

Un environnement complet de développement de simulation devrait évidemment essayer de combiner ces diverses méthodes pour en tirer le meilleur. Nous nous intéressons uniquement à la dernière méthode citée: la répartition des processus du modèle.

2.6.5 Caractéristiques de la simulation répartie

Outre la rapidité potentielle de la simulation répartie par rapport à la simulation séquentielle classique (grâce essentiellement à la coopération entre les *PL*s ou les machines qui les supportent), la simulation répartie possède les caractéristiques suivantes:

- La simulation peut être adaptée à la configuration matérielle disponible. Le placement des *PL*s sur chaque machine dépend de l'interaction ou du degré de communication entre les *PL*s. Deux *PL*s communicants régulièrement doivent être simulés sur la même machine (ou le même site) pour diminuer les temps de communication.
- Chaque *PL* est totalement indépendant et la communication entre eux s'effectue par échange de messages en mémoire ou sur le réseau d'interconnexion entre les processeurs. Le contrôle de l'ensemble se fait de manière distribuée.
- Chaque *PL*, grâce à son indépendance, a ses données locales, non partagées avec les autres.

2.7 Contexte distribué

Le support d'exécution de la simulation est constitué par un ensemble de processeurs asynchrones entre lesquels circule de l'information sous forme de messages. Le transfert d'un message d'un processeur à un autre prend un temps fini mais arbitraire. Il faut alors résoudre deux problèmes pour réaliser une simulation distribuée sur une telle machine:

- Le placement des processus logiques sur les processeurs de la machine.
- La construction d'un schéma d'exécution qui, d'une part, prend en compte du temps simulé pour garantir sa *progression* et, d'autre part, assure que les processus logiques sont exécutés correctement par rapport au temps simulé (pour respecter les *contraintes de causalité* locales).

Le problème de placement n'est pas abordé ici. Le lecteur peut se rapporter à [FR87] (chap. 5). Toutefois, notons que la création dynamique de processus peut être nécessaire dans un programme de simulation et que, par conséquent, l'algorithme de placement et d'ordonnancement des tâches doit en tenir compte.

Le second problème concerne la répartition du temps simulé global. On affecte pour cela une horloge (logique) locale à chaque processus logique; le rôle de celle-ci est de représenter l'écoulement du temps simulé global. Afin que l'ensemble de ces horloges locales représente correctement le temps simulé, il est nécessaire de synchroniser leurs évolutions. Deux classes de solutions sont possibles et se distinguent par la synchronisation plus ou moins forte qu'elles imposent à l'évolution des horloges. Il s'agit des versions réparties de la simulation, dirigées respectivement par le temps et par les événements.

2.8 Relation et contrainte de causalité

2.8.1 Relation de causalité entre événements

Définition 2 (relation de causalité entre deux événements)
Un événement A est une cause d'un événement B s'il existe une suite d'événement $e_0 := A, e_1, \ldots, B := e_n$, telle que pour chaque paire (e_i, e_{i+1}) d'événements adjacents:

1. *e_i et e_{i+1} sont dans le même processus et la date d'exécution de e_i est plus petite que celle de e_{i+1}; ou*
2. *l'événement e_i envoie un message d'événement à e_{i+1} qui le recevra dans un délai fini.*

Cette relation correspond à la relation "*happens before*" de Lamport [Lam78]. Intuitivement, la contrainte majeure dans la mise en œuvre des dates de traitement peut se résumer au lemme suivant:

Lemme 1
Si un événement A est la cause d'un événement B, alors le traitement de A et de B doivent être planifiés en temps réel de manière à ce que A termine complètement son traitement avant que B ne commence le sien.

2.8.2 Contrainte de causalité

Définition 3 (contrainte de causalité locale)
La contrainte de causalité locale est dite satisfaite par un processus logique si, et seulement si, les messages d'événement reçus par ce processus logique sont traités dans l'ordre non décroissant des estampilles.

Il y a une différence entre l'ordre chronologique des événements et les relations de causalité entre événements. La *relation de causalité* impose un ordre partiel d'exécution sur les événements. Si e_1 précède e_2 au sens de cette relation , e_1 est directement ou indirectement la *cause* de e_2; la simulation de e_1 devra alors précéder celle de e_2. Il n'est pas exclu que deux événements survenant à la même date soient liés par une relation de causalité: c'est le cas par exemple d'un PL qui, recevant un message à t, réagit en émettant instantanément (dans le même temps simulé) un autre message. Par contre, si deux événements ne sont pas liés par une relation de causalité, leur simulation peut se faire dans un ordre quelconque. Ainsi, ***la condition pour que l'on puisse exécuter deux événements*** e_1 ***et*** e_2 ***en parallèle est que*** e_1 ***et*** e_2 ***ne soient pas liés par une relation de causalité, et non pas que*** e_1 ***et*** e_2 ***portent la même date.***

Mais comment pourrait-on savoir que e_1 et e_2 ne sont pas liés par une relation de causalité, sans les avoir exécutés? Cette question est le problème fondamental que toutes les stratégies de simulation répartie tentent de résoudre.

A priori, on n'a aucune information sur la dépendance des événements. Alors on considère que les seuls événements sûrs à exécuter sont ceux dont l'estampille est la plus petite. Ainsi, l'exécution parallèle est équivalente à l'exécution séquentielle. Dans cette perspective, les relations de causalité dans le système physique deviennent une chaîne de contraintes de causalité locale dans le simulateur.

Lemme 2
Si tous les processus logiques satisfont la contrainte de causalité locale, et s'ils n'interagissent que par échanges de messages estampillés, alors la simulation dans son ensemble respecte la relation de causalité [Fuj90c].

Remarque 3
Le respect de la contrainte de causalité locale est une condition suffisante, mais pas nécessaire, pour garantir qu'aucune erreur de causalité ne puisse survenir dans l'ensemble du système. Car il se peut que deux événements sur deux PLs distincts soient totalement indépendants l'un de l'autre. Dans ce cas, leur traitement peut violer la contrainte de la causalité locale sans violer celle de la causalité globale.

Afin de garantir le respect de la contrainte de causalité dans un simulateur réparti, deux classes de solutions ont été proposées:

- La stratégie optimiste dans laquelle les messages sont traités dans l'ordre d'arrivée jusqu'à ce que l'on détecte une violation de la contrainte de causalité locale. Dans ce cas, il s'agit d'une erreur de synchronisation qui nécessite un retour en arrière dont le mécanisme est basé sur la notion de temps virtuel [Jef85].

- La stratégie pessimiste , dite aussi sûre ("conservative"), dans laquelle on met en œuvre un algorithme de la simulation assurant toujours le respect de la contrainte de causalité locale [CM79a].

2.9 Conclusion

En résumé, la simulation répartie est difficile car les contraintes imposent un ordre relatif dans lequel les calculs doivent s'exécuter. Cet ordre, en général assez complexe, dépend fortement de l'état des variables de chaque composante du modèle. La simulation répartie se distingue des autres domaines où le parallélisme a un grand succès par la nécessité de respecter la contrainte de causalité. La nature dynamique des problèmes de simulation répartie est la raison principale pour laquelle une solution générale est insaisissable.

Le simulateur réparti présente l'inconvénient d'exiger plus de mémoire que la simulation séquentielle. Par contre, il permet une modélisation assez proche de la réalité. Il se réalise aussi bien au niveau logiciel qu'au niveau matériel, en remplaçant dans le raisonnement précédent un processus logique par un processeur dans un environnement multiprocesseur.

Chapitre 3

Simulation asynchrone sûre

Introduction

Dans la simulation sûre, un processus logique s'engage à traiter un message d'événement (m, t) lorsqu'il est sûr qu'aucun autre message d'événement dont l'estampille est inférieure à t puisse lui parvenir. En désignant le $i^{ième}$ PL par PL_i et son horloge locale HL_i, les approches sûres garantissent le principe suivant:

Principe 3
$\forall PL_i, PL_j$, *si* PL_i *envoie un message à* PL_j *alors* $HL_i \geq HL_j$.

Un processus logique PL ne reçoit jamais un *message désynchronisé* (l'estampille du message reçu n'est jamais inférieure à l'horloge locale du PL). Par conséquent, HL_i ne recule jamais. L'horloge locale est dite ***monotone croissante***.

Les approches sûres traitent uniquement les messages d'événements de manière sûre. *Un message d'événement* (m, t) *est traité de manière sûre* par un PL si on peut assurer qu'aucun message d'événement (m', t') ne sera reçu ultérieurement par PL avec $t' < t$. Ainsi, on peut s'assurer que le traitement de (m, t) ne viole pas la contrainte de causalité. Un message d'événement traité de manière sûre est un ***message d'événement sûr*** que nous désignons simplement par ***événement sûr*** ou encore ***message sûr***.

La section 1 présente les opérations de base d'un processus logique au cours desquelles une situation d'impasse peut survenir. La section 2 présente les différents cas d'impasse. La section 3 présente les mécanismes de base de la résolution d'impasse; la section 4 les autres approches de résolution d'impasse. La section 5 présente une analyse des algorithmes sûrs.

3.1 Opérations d'un PL

On suppose que les canaux reliant les PLs sont traités comme des files d'attente PAPS (premier arrivé, premier servi) et sans perte de message. D'après les règles ci-dessus, les messages transmis par un canal arrivent au destinataire dans l'ordre chronologique des estampilles. Le principe 3 est respecté en appliquant les deux règles d'attentes suivantes:

R1: *règle d'attente en entrée ou règle d'attente à la réception de messages* ("Input waiting rule"):

un PL attend l'arrivée de messages dans tous ses canaux d'entrée avant de sélectionner le message d'événement à traiter. Le message choisi est celui dont l'estampille est la plus petite.

R2: *règle d'attente en sortie ou règle d'expédition de messages* ("Output waiting rule"):

un PL n'envoie aucun message (m', t') avec $t' \leq HL$ sur ses canaux de sortie. Autrement dit, chaque PL envoie, dans ses canaux de sortie, uniquement des messages dont l'estampille est supérieure à son horloge locale.

Entre la réception et l'émission d'un message, le processus passe par une phase de calcul interne (ses propres activités) de durée finie dans laquelle il met d'abord à jour son horloge locale à l'estampille du message reçu et traite le message, puis il détermine l'ensemble des canaux d'entrée sur lesquels il doit appliquer les règles d'attente pour son prochain traitement. Tous les messages d'événement (m, t) reçus ou émis par un PL doivent avoir $t \geq HL$.

3.2 Problème d'impasse

D'après le principe ci-dessus, les PLs ne possédant pas d'événement sûr sont bloqués et cela peut conduire à un interblocage (impasse).

Définition 4 (Impasse ou interblocage)
Un ensemble D de processus logiques PLs est dans une situation d'impasse ou d'interblocage en un instant de la simulation s'ils remplissent les conditions suivantes:

- $\forall PL \in D$, *soit PL attend un message, soit PL a terminé;*
- $\exists PL \in D$, *tel que PL attend la réception d'un message;*
- $\exists PL_i \in D$, *PL_i attend un message d'un autre PL_j de D $(i \neq j)$, et il n'y pas de message dans le canal reliant PL_j à PL_i.*

Il s'ensuit qu'aucun PL de D ne pourra continuer son exécution car les PLs s'attendent les uns les autres.

Un interblocage [CHM79] entre les processus peut survenir dans le réseau simulé, si il existe:

1. soit une branche du réseau de PL (c'est-à-dire qu'une partie du réseau) qui ne reçoit jamais de message. Le PL concerné se bloque en attente d'un message provenant par ce canal d'entrée: c'est la *famine* du PL.
2. soit des PLs forment un *circuit*; un canal d'entrée de l'un de ces PLs peut dépendre de l'un de ses propres canaux de sortie; auquel cas, il est bloqué indéfiniment.

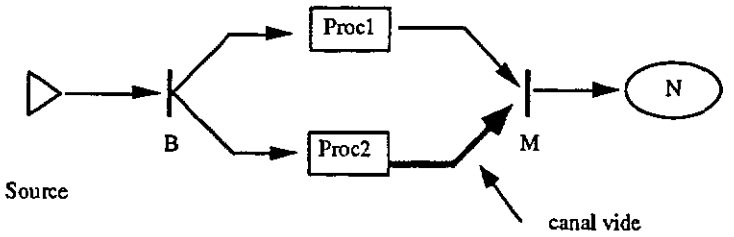


Figure 3.1: Simulation répartie ne pouvant pas progresser (famine)

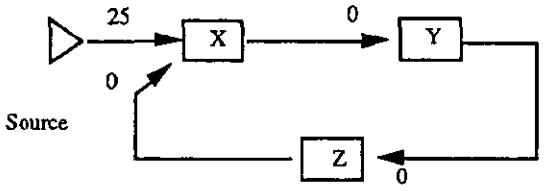


Figure 3.2: Simulation répartie dans un circuit

Avant d'illustrer les deux cas d'impasse dans le réseau simulé, donnons la définition de la date d'un canal.

Définition 5 (Date d'un canal)
Chaque PL détermine sur ses canaux une date définie comme suit:

- *elle est initialisée à 0;*
- *elle est égale à la plus petite estampille des messages en transit dans le canal*[1]*, si le canal n'est pas vide; sinon à l'estampille du dernier message ayant transité dans le canal.*

Exemple 1 (famine dans le réseau simulé)
Soit un modèle logique dont les messages sont envoyés de la source au PL B, puis de B aux PL Proc1 ou Proc2, de ceux-ci vers le PL M et ensuite sortent du réseau par le PL N. Le graphe de transmission des messages est donné dans la figure 3.1

Considérons le cas où B envoie tous les messages vers Proc1. Au cours de la simulation le PL M ne recevra jamais de message du PL Proc2. Une conséquence immédiate est que la date du canal reliant Proc2 à M reste à 0, ainsi M n'enverra jamais de message à N. Donc N ne recevra jamais de message de M, tandis que dans la réalité le modèle fonctionne normalement. (Rappelons que, grâce au principe de prédictabilité, le modèle peut toujours évoluer).

Exemple 2 (circuit dans le système)
Soit un système composé de trois processus logiques et d'une source dont le graphe de transmission de messages est donné par la figure 3.2.

Le nombre qui accompagne chaque flèche représente la valeur de la date du canal à un instant t de la simulation. En effet, nous sommes dans une situation où tous les canaux

[1] le message en tête du canal sous l'hypothèse PAPS

sont vides, sauf le canal reliant la source au processus logique X. Alors, à l'instant t, aucun processus X, Y ou Z ne peut envoyer de messages sans en recevoir d'abord.

Nous sommes dans un état où Z ne pourra pas envoyer de message à moins que X n'envoie premièrement de message à Y. Mais, dans la réalité, X n'a pas besoin d'attendre la réception d'un message de Z car il pourrait traiter le prochain message de la source. Comme un PL n'a qu'une connaissance locale du comportement de son PP, et non une vision globale du système, X ne pourra continuer sans qu'il ait reçu un message au moins de Z. Z, quant à lui, attend un message de Y pour continuer et Y, à son tour, attend un message de X. Ce qui conduit les processus X, Y et Z dans l'impasse.

3.3 Résolution de l'impasse

Deux solutions classiques permettent de traiter les impasses:

- **la prévention**: dans ce cas, un algorithme de contrôle vient s'adjoindre au simulateur pour éviter l'apparition de la situation d'impasse. Elle est basée sur l'émission de messages supplémentaires *(messages NULL)* [Mis86] [PLM91].
- **la détection et la guérison**: dans ce cas, le simulateur progresse tant que cela est possible en respectant les contraintes de causalité. Lorsqu'il n'est plus possible de progresser, un algorithme réparti détecte l'impasse et la résout [Cha82], [Nev90].

3.3.1 Prévention des impasses

La prévention d'impasse, appelée aussi mécanisme des messages nuls [CHM79] a été le protocole sûr le plus étudié. L'idée de base est de faire émettre par les PLs, en plus des messages de simulation, des messages de contrôle (nul, t) estampillés par la date t de leur émission. Ces messages de contrôle ne correspondent à aucun message du modèle simulé. Leur traitement par le PL récepteur permet de faire progresser sa date locale mais n'implique aucun changement de son état.

La preuve formelle de l'exactitude de la prévention d'impasse est présentée dans [Mis86] sous les conditions suivantes: l'environnement de réalisation doit garantir la stratégie PAPS des canaux et les liens de communication entre les processus logiques doivent être statiques.

Messages nuls simples

Une solution pour prévenir la famine consiste à émettre un message sur chaque canal de sortie d'un PL, lors du traitement d'un événement. Le processus PL_i, envoie au moins un message portant sa date courante sur chacun de ses canaux de sortie: un message normal (m, HL_i) pour les uns, s'il y en a à envoyer, et un message nul (nul, HL_i) pour les autres. Malheureusement, cette solution ne suffit pas pour exclure l'impasse dans la cas où il y a un circuit dans le réseau.

Messages nuls avec prévision

Lorsque la topologie du modèle contient un circuit, le mécanisme des messages nuls simples [CHM79] ne suffit plus pour éviter l'impasse. Pour la résoudre, il faut une information supplémentaire: la *prévision* ou *prédiction* δ ($\delta > 0$) pour faire avancer la simulation. Les messages nuls envoyés par un PL portent sa date courante augmentée de la prédiction: $(nul, HL + \delta)$. Dans ces conditions, le PL n'émettra aucun message avant $HL + \delta$. Pour un PL, la valeur de δ matérialise une prévision (ou prédiction) sur son comportement futur ("look ahead"). Il est toujours possible de déterminer une valeur delta qui débloque l'impasse, grâce à la propriété de prédicabilité du modèle. Ceci permet de faire progresser la date des PLs dans un circuit.

Variantes de la prévention des impasses à l'aide de messages nuls

Des variantes de la prévention d'impasse ont été proposées dans la littérature. Elles tentent toutes de diminuer la quantité de messages nuls circulant en vue d'accroître l'efficacité du simulateur.

Misra [Mis86] propose des envois de messages nuls à la demande. Un processus PL_i en envoie sur le canal (i, j) uniquement lorsque le successeur PL_j bloqué par l'absence de message sur ce canal en fait la demande. Si PL_i est capable de satisfaire cette demande, c'est-à-dire qu'il peut envoyer un message (nul ou non) de date supérieure à celle du canal, PL_j sera débloqué. Autrement PL_i va à son tour envoyer une demande à un ou plusieurs de ses prédécesseurs. Ainsi, les demandes peuvent être cycliques et engendrer elles-mêmes des impasses. Cette impasse peut être résolue par le premier processus initiateur de la demande cyclique, étant donné que le chemin parcouru par une demande (processus visités et dates des canaux visités) est toujours mémorisé par la demande. Si le nom du processus recevant la demande y apparaît plus d'une fois, alors l'impasse est détectée et la phase de résolution lancée. Cette technique, qui n'a pas de suite dans la littérature, permet de diminuer la quantité de messages circulant, mais augmente le temps de synchronisation car il faut au minimum un aller-retour de message pour qu'un processus obtienne l'avancement de la date d'un canal.

Le rôle des messages nuls dans la simulation est étendu dans la variante de l'algorithme initial de Chandy et Misra présentée par Cai et Turner dans [CT89]. Ils véhiculent les dates de prédiction mais aussi des informations sur le système. Un message nul mémorise le chemin qu'il a parcouru. Ceci permet à un processus de connaître les dépendances interprocessus et les circuits se trouvant dans le modèle. Il porte également l'estampille du premier message normal à traiter dans le chemin parcouru. Cette estampille, qui est la date de prédiction optimale dans le circuit, permet de limiter le nombre de messages nuls circulant. Selon les résultats expérimentaux obtenus par Cai et Turner, cette technique est plus efficace que le mécanisme des messages nuls simple pour certaines classes de réseaux de files d'attente. La prédiction constitue généralement le temps de service dans les réseaux de files d'attente.

Su et Seitz [SS89] proposent différentes variantes de l'algorithme des messages nuls simples et de l'algorithme des messages nuls à la demande. Le but de ces variantes, qui peuvent être adaptatives selon le modèle simulé, est de trouver un compromis entre le

nombre de messages engendrés (nuls ou non) et la résolution efficace des impasses. L'envoi d'un message nul peut être différé de manière "intelligente", c'est-à-dire en évitant tout risque d'impasse, dans l'espoir qu'au prochain envoi, l'estampille portée par ce message augmente. Un seul message remplacera ainsi plusieurs autres messages. De la même manière, deux ou plusieurs messages normaux peuvent être regroupés dans un même message pour limiter le trafic. Ces deux moyens de regroupement de messages peuvent être combinés pour former d'autres variantes. L'algorithme des messages nuls à la demande peut être amélioré en associant un temps maximal d'attente à la sortie d'un processus. Ainsi, si la demande n'arrive pas avant l'écoulement de ce temps maximal, un message nul est envoyé.

Mehl introduit du risque dans le protocole sûr [Meh91]. Les périodes d'inactivité des processus sont utilisées pour traiter à l'avance d'autres événements. Ce traitement n'affecte pas les états et les événements corrects mais permet de gagner du temps par la suite dans le cas où ses résultats sont corrects.

3.3.2 Détection-Guérison des impasses

Un autre protocole de gestion des impasses a été introduit par Chandy et Misra [ChandyM81]. L'idée de cette solution consiste à répéter la séquence suivante:

1. exécution jusqu'à l'impasse.
2. détection de l'impasse.
3. guérison de l'impasse en relançant l'exécution de un ou plusieurs *PLs*.

Lors de la première phase, les processus logiques n'émettent que des messages d'événement selon le principe de message sûr. Ils font progresser la simulation en traitant à chaque cycle le message ayant la plus petite estampille sur ses canaux d'entrée. Une impasse survient lorsque tous les *PLs* sont bloqués en attente d'un message provenant d'un canal d'entrée vide ou en attente de pouvoir envoyer un message sur un canal de sortie (cas où le canal peut être une mémoire-tampon de taille limitée).

Un processus spécial de contrôle (ou contrôleur) détecte l'impasse grâce à un algorithme réparti. Par exemple, celui de Dijkstra et Scholten [DS80] ou celui de Chandy et Misra [Cha82].

Une fois l'impasse détectée, le contrôleur lance un calcul qui permet de déterminer le ou les *PLs* pouvant redémarrer, sans violer la contrainte de causalité. La simulation reprend son déroulement normal et reprend la première phase.

Remarque 4
Il faut noter qu'il existe toujours au moins un *PL* pouvant reprendre la simulation si l'on fait l'hypothèse que le modèle et le programme de simulation correspondant sont corrects. En effet, parmi tous les messages en attente sur les canaux d'entrée des processus, celui qui a la plus petite estampille t_{min} peut être traité par son destinataire car plus rien ne peut modifier l'état du système avant la date t_{min}. L'algorithme de guérison consiste alors simplement à déterminer les messages estampillés par t_{min} et leurs destinataires.

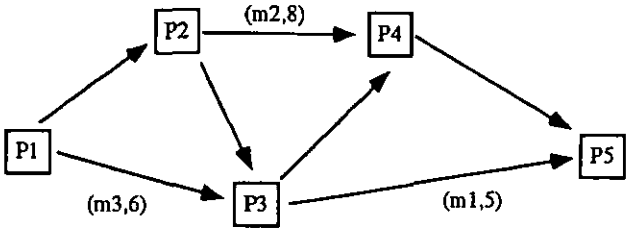


Figure 3.3: Réseau simulé pouvant conduire à une famine

Ingels et Raynal [IR90] améliorent cet algorithme en admettant que certains messages peuvent être envoyés même s'ils ont une date ultérieure à la borne inférieure. En effet, parmi les messages en attente de traitement sur la figure 3.3, $(m1, 5)$ peut être envoyé (de date minimale), mais aussi $(m3, 6)$ car le prédécesseur $PL2$ qui bloque $PL3$ est déjà à la date 8. Ceci n'est pas le cas de $(m2, 8)$ car $PL4$ peut recevoir un message de date antérieure à 8 provenant de $PL3$. Dans le cas général, un message (m, t) peut être envoyé par un processus PL_i si l'estampille t du message est antérieure aux estampilles de tous les messages déjà envoyés par les processus en amont de PL_i (processus PL_a tel qu'il existe un chemin de PL_a vers PL_i). Cet algorithme amélioré est décrit dans [IR90], pour le cas où les mémoire-tampons des canaux d'entrée d'un processus sont de taille infinie, et dans [RF87], pour le cas de celles de taille limitée.

3.4 Autres approches de résolution de l'impasse

Diverses approches de la *synchronisation des opérations* [CS89b], [Lub89], [Nic88] ont été proposées pour éviter le problème de l'impasse. Ces approches déterminent itérativement les événements sûrs à traiter, puis les traitent. Des barrières de synchronisation sont utilisées pour maintenir la cohérence des itérations lorsque les événements interférent entre eux. Ces algorithmes sont mieux adaptés aux architectures à mémoire partagée. La détection de l'impasse est implicitement résolue (transportée intrinsèquement) dans les ordinateurs à mémoire partagée. Passons brièvement en revue ces approches:

L'*approche sûre de la fenêtre temporelle* [Lub89] ("conservative time windows"). Le but de la fenêtre temporelle est de réduire la recherche de l'événement sûr à traiter dans un intervalle de temps (ou fenêtre temporelle). La borne inférieure de la fenêtre comporte la date minimale des évenements non-traités. Seuls les évenements situés dans la fenêtre sont éligibles pour le prochain traitement. Une question importante est la détermination de la taille de la fenêtre temporelle. La taille appropriée d'une application spécifique exige des informations soit du programmeur, soit du compilateur ou du contrôle de la simulation en cours d'exécution.

L'*amélioration de la prévision* ("lookahead") en précalculant les temps de service [Nic88], "unité d'avance ou prévison", se rapporte à la possibilité de prédire ce qui se passera, ou mieux, à ce qui ne se produira pas dans le futur simulé. Si un processus au temps simulé $Tsim$ peut prédire parfaitement tous ses événements qu'il engendrera jusqu'au temps simulé $HL + \delta t$, alors la prévision du processus est δt avec $\delta t > 0$. L'utilisation des prévisions augmente la prédiction des événements futurs qui, en retour, sert à

déterminer d'autres événements certains à exécuter. L'aptitude d'estimation des calculs d'avance exige des applications spécifiques la capacité de prédire les unités d'avance. La prévision des événements futurs est une technique puissante quand elle peut être appliquée.

La *connaissance conditionnelle* ("conditional knowledge") [CS89b]. Dans le simulateur, on distingue deux types d'événements: les événements définis et les évenements conditionnels. Les événements conditionnels surviennent si un prédicat est satisfait, alors que les autres surviennent sans condition. Le simulateur est toujours certain d'exécuter les évenements définis mais il n'en est pas de même pour les événements conditionnels. Cette stratégie utilise un protocole de simulation sûre parallèle où la connaissance conditionnelle est utilisée pour déterminer les événements sûrs.

L'*exploration de la topologie du réseau* [Kum89], [Nev89], [De 90] [LLB90]. Cette stratégie exploite les propriétés topologiques du réseau pour rendre plus efficaces les algorithmes de simulation.

3.5 Analyse des techniques de synchronisation sûre

L'inconvénient manifeste des approches sûres est qu'elles n'exploitent pas pleinement les potentialités du parallélisme dans l'application de la simulation. Elles sont toujours trop pessimistes et forcent l'exécution séquentielle quand cela n'est parfois pas nécessaire. Les algorithmes sûrs sont mal adaptés aux applications de simulation ne pouvant pas exploiter les propriétés de prévision, même si l'application comporte un grand degré de parallélisme.

Les méthodes sûres ne sont pas robustes car un changement mineur dans l'application peut avoir un effet catastrophique sur la performance du système.

Les techniques sûres nécessitent une configuration statique: on ne peut pas créer dynamiquement les processus logiques et les canaux orientés qui les relient. Ils sont définis et créés une fois pour toute au début de la simulation.

La plupart des schémas sûrs nécessitent que l'utilisateur fournisse des connaissances explicites sur le comportement des processus logiques pour définir leurs protocole de synchronisation. Seul l'algorithme de détection de l'impasse et de rétablissement ne nécessite pas explicitement des connaissances de l'utilisateur.

Le défaut majeur des protocoles sûrs est que l'utilisateur ou le programmeur d'une application répartie doit s'occuper des détails du mécanisme de synchronisation afin d'obtenir une bonne performance.

Chapitre 4

Simulation asynchrone optimiste

Résumé

La distorsion temporelle ("time warp") [Jef85] est une méthode de synchronisation de la simulation optimiste. Elle n'impose aucune contrainte sur l'évolution asynchrone des processus de la simulation. Ainsi, elle admet la violation de la contrainte de causalité. Mais elle garantit, à la réception d'un message (m,t) dont l'estampille est antérieure à l'horloge locale du PL, le **message désynchronisé**, de pouvoir retourner en arrière pour rétablir le dernier état cohérent sans erreur de causalié avant t. Ce chapitre est entièrement consacré à l'étude de la distorsion temporelle en présentant d'abord brièvement le paradigme du temps virtuel.

Introduction

La ***distorsion temporelle*** ("time warp"), développée par Jefferson [Jef85] est basée sur la notion de temps virtuel. Elle n'impose aucune contrainte sur l'évolution asynchrone des processus de la simulation: chacun avance à son propre rythme et son horloge locale progresse en conséquence. Lors de l'arrivée d'un message (m,t), le processus logique récepteur fait progresser son horloge jusqu'à t et traite le message, ce qui peut provoquer des envois de messages estampillés avec des valeurs supérieures ou égales à t. La perte de synchronisation survient lorsque le processus récepteur reçoit un message désynchronisé (m,t). Il faut, dans ce cas, remonter l'horloge locale de la simulation jusqu'à t, puis traiter le message (m,t) et recommencer la simulation à partir de t. Ainsi, dans une simulation optimiste, chaque PL garantit à la réception d'un message désynchronisé (m,t) de remonter son temps local de simulation jusqu'au dernier état cohérent avant t et de recommencer la simulation à partir de ce point.

L'avantage majeur de cette approche est qu'elle permet d'exploiter pleinement le parallélisme du systéme, même dans des situations où la contrainte de causalité peut être violée. Cette approche se prête également à la création dynamique des processus logiques [TA89].

La section qui suit présente brièvement le paradigme de temps virtuel. La suivante

présente une étude approfondie du mécanisme de distorsion temporelle. Ce mécanisme comporte deux mécanismes de contrôle: l'un local, pour assurer les relations de causalité locale entre les voisins proches, et l'autre global, pour assurer la cohérence du simulateur entier. La dernière section présente une analyse critique des méthodes optimistes.

4.1 Paradigme de temps virtuel

Le temps virtuel est un paradigme pour organiser et synchroniser les systèmes répartis qui peuvent être appliqués aux problèmes de la simulation répartie par événements discrets. Un système à temps virtuel est un système réparti qui s'exécute en coordination avec une horloge virtuelle imaginaire qui avance. Le temps virtuel est une variable globale utilisée pour mesurer l'avancement de la simulation et pour synchroniser les PLs.

Les actions qui ont lieu dans un espace virtuel x au temps t sont représentées par un événement (x, t).

Les processus communiquent par échange de messages. Chaque processus est libre d'envoyer à un instant quelconque un message à un autre (y compris à lui-même). Il n'y a pas de "concept de canal" entre deux processus et on n'a besoin d'aucun protocole pour entrer dans un dialogue ou pour clore un dialogue.

Tous les messages émis possèdent au moins quatre valeurs: le nom de l'expéditeur, la date virtuelle d'expédition, le nom du receveur et la date virtuelle de réception. Un message transfère l'information d'un point d'événement à un autre dans l'espace-temps virtuel.

Un système virtuel est soumis à deux règles. D'une part, la date virtuelle d'expédition d'un message doit être inférieure à sa date virtuelle de réception. D'autre part, la date virtuelle de chaque événement d'un processus en cours de traitement doit être inférieure à la date virtuelle du prochain événement que le processus traitera.

Actions possibles d'un processus

Un événement (x, t) est un calcul séquentiel déterministe à l'intérieur d'un processus x pouvant poser les actions suivantes:

- x peut recevoir un nombre quelconque de messages (y compris de lui-même);
- il peut lire sa date virtuelle locale t;
- il peut mettre à jour ses variables d'état;
- il peut envoyer un nombre quelconque de messages dont l'expéditeur est automatiquement x et la date virtuelle d'expédition t.

4.2 Origine de la "distorsion temporelle"

Les fondements du mécanisme de distorsion temporelle furent inventés par Henry Sowinzral et David Jefferson pour accélérer les applications de la simulation répartie [JS85]. Ils proposent un outil de synchronisation de la simulation répartie qui permet de *défaire une action locale*: le rollback. Le **rollback** consiste à retourner au plus récent état cohérent (sans erreur de causalité) qu'on doit avoir sauvegardé. La réalisation simple et élégante de la synchronisation d'une simulation optimiste est basée sur la distorsion temporelle, où interviennent la notion d'*antimessage* (cf. section 4.4.2) et de *rollback* (cf. section 4.4.4).

Plus tard, la notion de temps virtuel fut introduite comme un paradigme dans l'organisation et la synchronisation des systèmes répartis à temps virtuel [Jef85]. Le temps virtuel représente alors un axe temporel de référence dans une application. Dès lors, le mécanisme de distorsion temporelle n'est plus uniquement appliqué à la simulation répartie mais, comme la réalisation primaire de synchronisation, à tous les systèmes à temps virtuels. Il y a une analogie entre la théorie de la mémoire virtuelle et celle du temps virtuel. Leurs réalisations ont recours respectivement au mécanisme de pagination et au mécanisme de distorsion temporelle.

4.2.1 Domaines d'application de la distorsion temporelle

La distorsion temporelle n'est pas adaptée à toute application répartie. Elle est mieux adaptée aux applications dont le comportement peut être spécifié dans un système à temps virtuel. Des protocoles de synchronisation plus simples que la distorsion temporelle fournissent parfois de meilleures performances sous certaines conditions. Nous pensons particulièrement à la synchronisation sûre qui s'adapte mieux à un réseau statique [1]. Malgré ces restrictions, le mécanisme de distorsion temporelle a un vaste champ d'application aux problèmes de "synchronisation de temps virtuels".

4.2.2 Problème de synchronisation des temps virtuels

Soit une application composée de processus qui communiquent par des messages estampillés par leur date virtuelle de traitement. Les messages reçus sont ordonnés dans chaque processus (selon la figure 4.1) dans une une liste unique par rapport à leurs estampilles. L'expéditeur spécifie au receveur l'ordre dans lequel les messages doivent être traités à l'aide de l'estampille. Les messages peuvent ne pas arriver chez le receveur dans l'ordre croissant des estampilles, bien que le mécanisme de distorsion temporelle fournisse une meilleure performance, lorsque cet ordre coincide avec l'ordre d'arrivée des messages.

Le problème de synchronisation des temps virtuels consiste à garantir que "chaque processus traite localement ses messages d'événement dans un ordre non-décroissant des estampilles?"

On résout ce problème en introduisant une primitive de synchronisation plus forte qui consiste à défaire une action locale: le rollback (retour arrière). A la réception d'un message désynchronisé (m, t) par un PL récepteur dont l'horloge virtuelle locale est LVT

[1] un réseau statique est un réseau dont le nombre de processus et le nombre de canaux sont statiques

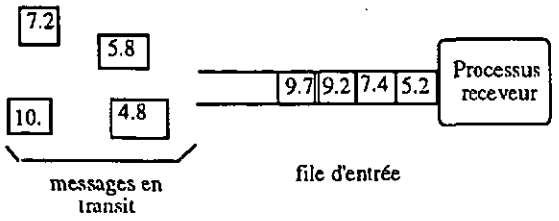


Figure 4.1: Un processus de synchronisation de temps de virtuels

("Local Virtual Time"), le processus fait revenir le simulateur entier à son dernier état global cohérent précédent t en effectuant les opérations suivantes:

1. retour au dernier état local cohérent du processus avant t.
2. rétablissement de l'état du simulateur entier (et non seulement du processus) avant t, en annulant toutes les actions effectuées entre t et LVT, par le PL lui-même et par les autres PLs, suite aux messages envoyés à ceux-ci dans cet intervalle de temps.
3. traitement du nouveau message d'événement (m, t).

En général, il est impossible de résoudre le problème de synchronisation en utilisant uniquement l'information locale si le seul outil de synchronisation autorisé est le blocage des processus. Mais on y arrive à l'aide de la primitive de synchronisation plus forte qui consiste à défaire une action locale: le rollback.

4.3 Mécanisme de distorsion temporelle

L'estampille d'un message est la date virtuelle de réception du message dans un système virtuel. Dans une simulation asynchrone optimiste, chaque processus garantit le principe suivant:

Principe 4 (Principe de la simulation optimiste)
Les messages d'événements (m, t) sont traités dans l'ordre d'arrivée, et le processus récepteur PL fait progresser son horloge virtuelle locale LVT à t. Si un message désynchronisé (m, t) arrive à PL, c'est-à-dire $t < LVT$, le PL retourne en arrière ("roll back") à la date t, annule toutes les actions locales et propagées entre t et LVT, et reprend normalement la simulation à partir de t.

Le terme "distorsion temporelle" provient du fait que, d'une part, les différentes horloges locales virtuelles des processus logiques ne sont pas synchronisées et, d'autre part, qu'elles peuvent avancer ou reculer dans le temps virtuel. La valeur de LVT n'est donc pas monotone à cause des retours en arrière possibles. En revanche, le système possède une variable monotone non-décroissante qui ne recule jamais: la *date virtuelle globale* ou GVT ("Global Virtual Time").

Un PL optimiste traite les messages d'événement sans respecter à priori la contrainte de causalité. Lorsqu'une erreur de causalité survient, violant la contrainte de causalité locale, il la détecte et la corrige. La causalité est donc respectée a posteriori.

Remarque 5
Les contraintes sur le modèle sûr et sur l'environnement d'exécution telles que la réalisabilité, la prédictabilité et le réseau statique que l'on rencontre dans la synchronisation sûre, n'existent plus dans la synchronisation optimiste.

Différents types de contrôles du mécanisme

Le mécanisme de distorsion temporelle [JBH+85], [Sow85] comporte deux parties principales: le contrôle local et le contrôle global. Le mécanisme de contrôle local assure que les événements sont exécutés dans un bon ordre et que les messages respectent la contrainte de causalité locale. Il fournit une "synchronisation locale" du simulateur. Le contrôle global s'occupe de la cohérence globale du système par la gestion de la mémoire, du contrôle de flux des messages, des entrées et sorties (E/S), des erreurs et de la détection de la terminaison. Il assure la synchronisation globale du simulateur. Il se base sur l'horloge globale virtuelle. Le mécanisme de distorsion temporelle est défini indépendamment de l'architecture des ordinateurs.

4.4 Contrôle local du simulateur

Les activités synchrones réelles du modèle simulé (système physique) sont maîtrisées en incluant la date de traitement dans chaque message transmis d'un processus logique à un autre. Ainsi, les processus logiques peuvent s'exécuter parallèlement aussi longtemps que leurs contreparties dans le système physique s'exécutent indépendamment. Chaque processus possède une horloge virtuelle locale LVT. La valeur de LVT ne change pas au cours du traitement d'un événement mais uniquement à la prise en charge d'un nouveau message d'événement. Le traitement du message devient délicat quand le processus reçoit un message désynchronisé.

Pour mieux comprendre le mécanisme de rollback, nous décrivons la structure des processus et des messages.

4.4.1 Description d'un PL

Un PL optimiste (cf. la figure 4.2) est décrit par:

- le **nom** du PL pour l'identifier;
- l'horloge virtuelle locale (LVT) qui indique localement l'avancement du temps simulé du processus;
- l'**état** ou l'ensemble des variables du PL, y compris sa pile d'exécution;
- la **liste des messages en entrée** ("*inputQueue*") qui contient la liste des messages reçus ordonnés selon leur estampille. Certains messages sont déjà traités. Néanmoins, ils ne sont pas effacés de la liste, si leur estampille est supérieure à GVT;

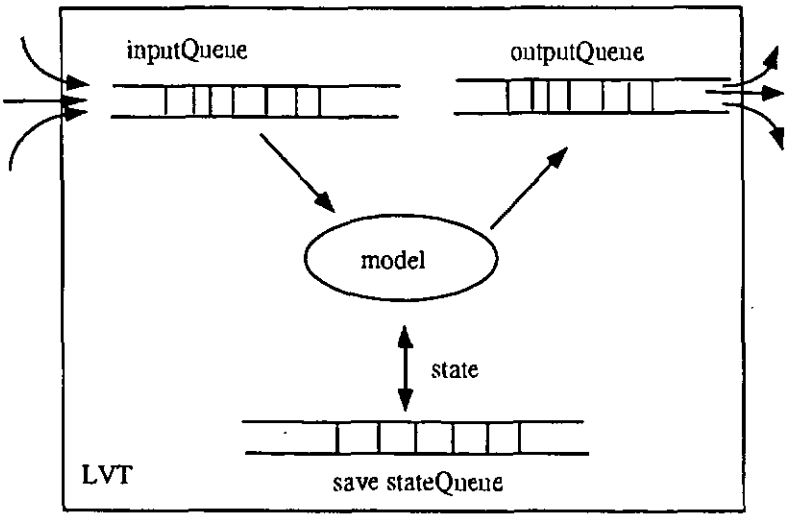


Figure 4.2: Description d'un *PL* optmiste

- la **liste de messages en sortie** (*"outputQueue"*) qui contient la copie des messages que le processus a récemment envoyés. Ils sont mémorisés dans l'ordre croissant de leur date virtuelle d'expédition. Ces messages sont appelés **antimessages** car ils sont utilisés pour annuler des messages déjà envoyés; ils se distinguent des autres messages par un attribut spécifique (positif, négatif). Comme dans la liste des messages d'entrée, seuls les messages dont la date virtuelle d'expédition est supérieure à GVT, ne sont pas effacés;

- la **liste des états sauvegardés** (*"stateQueue"*) contient la copie des récents états sauvés par le processus en vue de rendre possible l'exécution de la procédure de rollbak. Seuls les états dont l'estampille est supérieure ou égale à GVT sont retenus dans la liste. Comme nous le verrons dans le contrôle global, il n'est pas nécessaire de retenir les états auxquels jamais on ne retournera;

- la **liste des opérations définitives** (*"safeQueue"*) qui contient les opérations qui ne seront plus remises en cause (E/S par exemple) et qui sont donc effectuées uniquement lorsqu'elles ne risquent plus d'être défaites par un rollback. Il s'agit des états dont l'estampille est inférieure à GVT. C'est la sous-liste des états fossiles de la liste des états sauvegardés.

4.4.2 Message et Antimessage

Un message échangé entre deux processus porte dans la simulation optimiste des informations supplémentaires par rapport à un message dans la simulation sûre. La structure d'un message est:

$< m, exp, dst, t_{exp}, t_{dest}, s >$ où:

- m est l'identificateur de l'événement comme dans le cas du protocole pessimiste;

- t_{exp} est l'heure d'envoi du message (LVT de l'expéditeur).
- t_{dest} est l'heure de traitement du message par le récepteur du message. C'est l'estampille du message d'événement;
- *exp*: est l'identificateur de l'expéditeur;
- *dst*: est l'identificateur du récepteur du message.
- s est le signe du message: + ou bien −.

A chaque message correspond un *antimessage* de contenu identique mais de signe opposé. Deux messages identiques au signe près sont appelés ***antimessages*** l'un de l'autre.

Les messages du modèle simulé sont de signe positif (m^+); leurs antimessages ont un signe négatif (m^-). Quand un processus envoie un message, le message est introduit dans la liste des messages en entrée du récepteur et l'antimessage dans sa liste de messages de sortie.

L'utilité des antimessages réside aux propriétés du lemme suivant:

Lemme 3
Pour toute liste de messages L et pour tout message m,

- $(m^-)^- =$ m.

  l'anti-antimessage d'un message est égal à lui-même.
- Insert(Insert(L, m), m^-) = Insert(Insert(L, m^-), m) = L.

  quand un message et son antimessage surviennent dans une même liste de messages, ils s'annulent mutuellement.

La manipulation des messages négatifs est symétrique à celle des messages positifs. En particulier, lorsqu'un message négatif est émis, son antimessage (positif) est introduit dans la liste de sortie; c'est le message d'événement qu'il faudra retraiter une fois la synchronisation rétablie.

En général, les messages et antimessages sont créés par paire et s'annulent par paire; ainsi, à un instant donné, la somme algébrique de tous les messages dans un simulateur optimiste est nulle.

4.4.3 Rollback et mécanisme d'annulation

L'exécution d'un PL optimiste est un cycle dans lequel il reçoit des messages d'événements et les exécute dans l'ordre croissant de leur estampille. Quand un processus a fini de traiter les messages de sa liste en entrée, *inputQueue*, il attribue à son LVT la valeur $+\infty$; par convention, on dit qu'il est terminé. Mais le processus n'est pas détruit, ni son passé. Car l'arrivée d'un éventuel message désynchronisé peut faire retourner en arriére son LVT et défaire les actions locales.

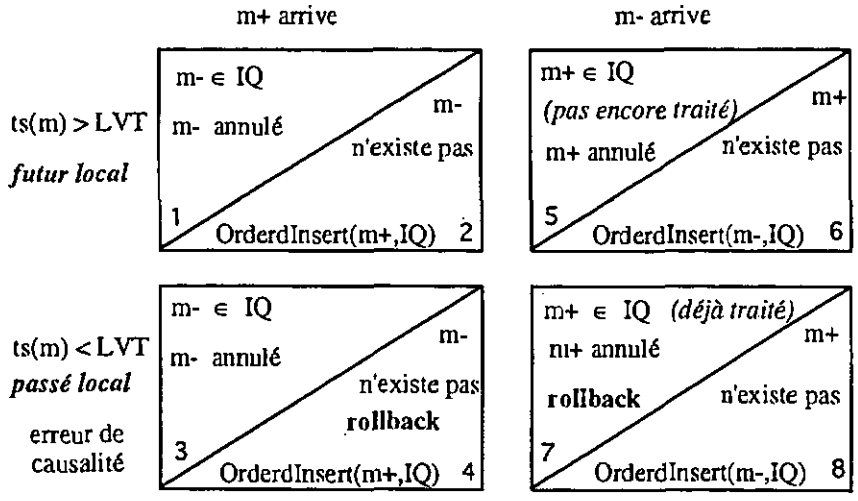


Figure 4.3: Mécanisme de synchronisation des messages

Le mécanisme de synchronisation (cf. la figure 4.3) agit en fonction du message reçu et de la valeur de l'horloge locale LVT. Un message d'événement m dont l'estampille $ts(m)$ est supérieure ou égale à LVT est dans le ***futur local*** du PL. Un message d'événement m dont l'estampille $ts(m)$ est inférieure à LVT est dans le ***passé local*** du PL. Ce message d'événement est un ***message désynchronisé***. Le traitement d'un message estampillé dans le passé local du processus est un indicateur d'erreur de causalité ou de violation de la contrainte de causalité locale. Le mécanisme du rollback restaure le plus récent état cohérent d'après sa liste des états sauvés *stateQueue*. Un état cohérent (stable ou correct) est un état satisfaisant de la contrainte de causalité. Les effets externes des messages d'événement traités, estampillés dans le passé local, sont annulés en envoyant leurs antimessages dans *outputQueue*.

4.4.4 Retour au dernier état cohérent du système

Le rétablissement de l'état correct du simulateur se fait par le mécanisme du rollback qui comporte trois phases:

P1: Le retour à l'état local cohérent avant $ts(m)$

Le retour à un état local cohérent du processus consiste à:

1. la recherche dans la liste des états sauvés *stateQueue* du dernier état sauvé avant $ts(m)$ et sa restauration.
2. le retrait de *stateQueue* des états sauvés après $ts(m)$.
3. la mise à jour de LVT (l'horloge virtuelle locale) du processus.

P2: L'annulation des actions externes entre $ts(m)$ et LVT

Toutes les actions externes entreprises par les événements qui étaient planifiés entre $ts(m)$ et LVT sont annulés en appliquant le lemme suivant [Jef85]:

Lemme 4
Pour annuler un message déjà envoyé, il suffit simplement de transmettre son antimessage.

Ainsi, le processus envoie à ses récepteurs les antimessages de *outputQueue* dont la date virtuelle d'expédition est entre $ts(m)$ et LVT. Ces antimessages vont annuler les actions des messages positifs qui auraient été transmis.

P3: **La reprise de la simulation.**

A cet instant, l'ensemble du simulateur a un comportement (variables locales, listes de messages, horloges virtuelles locales et état locaux) cohérent jusqu'au temps $ts(m)$. C'est comme si les autres messages entre $ts(m)$ et LVT n'avaient jamais existé. Maintenant, il procéde au traitement du message désynchronisé.

Remarque 6
La première phase P1 sert à établir l'état local cohérent du processus récepteur du message désynchronisé alors que la seconde P2 sert à établir l'état cohérent des processus destinataires (successeurs de PL). Ainsi, après ces phases, l'ensemble du simulateur est dans un état global cohérent.

Rétablissement de l'état cohérent local

On n'est pas obligé de conserver l'état des processus après chaque événement. On peut aussi se contenter de points de reprise plus ou moins espacés. Évidemment, plus les points de reprise sont espacés, plus un rollback est coûteux en temps d'exécution, car il faut "défaire", puis "resimuler", sur une plus longue période. Le nombre d'états sauvés entre le dernier point de reprise et la date courante LVT représente la ***longueur du rollback***. Si, à un instant donné, la valeur de l'horloge virtuelle globales vaut GVT_0, il suffit de conserver les états (messages) passés jusqu'au dernier point de reprise précédant GVT_0. Dans la situation de la figure 4.4, un message désynchronisé estampillé td arrive alors que le temps simulé courant est tc, avec $tc > td$, provoquant un rollback à tr, date du plus récent point de reprise. Le rollback entre tc et td, et celui entre td et tr, sont traités différemment:

- Entre tc et td, le rollback doit permettre de tenir compte d'une évolution du système simulé différente de celle que l'on avait effectuée avant l'arrivée du message désynchronisé estampilé par td. Cette phase implique des actions externes, telles l'envoi d'antimessages qui peuvent à leur tour provoquer d'autres rollbacks.

- Entre td et tr par contre, il s'agit juste de recalculer les états non sauvegardés. L'évolution du système est la même que précédemment car le message désynchronisé ne peut avoir de conséquence sur l'évolution antérieure du système. Il ne faut donc ni réémettre les messages d'événement resimulés, ni envoyer d'antimessage. Cette phase, strictement interne, n'induit donc pas de rollback pour d'autres processus.

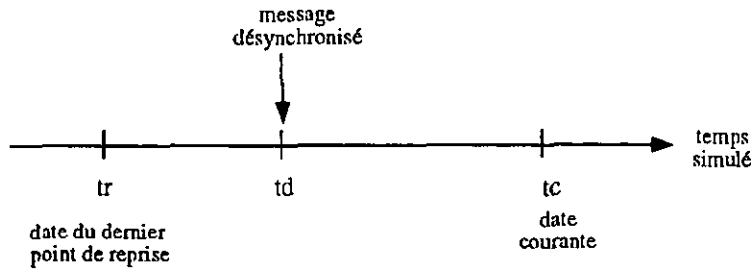


Figure 4.4: Description du rollback

Traitement à la réception d'un message positif m^+

Lors de la réception d'un message positif par un PL, deux cas peuvent se présenter: soit il est dans le futur local, ce qui correspond à une exécution normale, soit il est dans le passé local, ce qui peut entraîner un rollback.

- A la réception de m^+ dans le futur local, deux cas peuvent se présenter: soit son antimessage m^- est dans la liste des messages d'entrée (cas 1 de la figure 4.3) et ils s'annulent mutuellement; soit son antimessage n'existe pas (cas 2 de la figure 4.3) et m^+ est chronologiquement inséré dans *inputQueue*.

- A la réception de m^+ dans le passé local, deux cas peuvent se présenter: soit son antimessage m^- est dans la liste des messages d'entrée (cas 3 de la figure 4.3) et ils s'annulent mutuellement; soit son antimessage n'existe pas (cas 4 de la figure 4.3) et il faut retourner à l'état de l'ensemble du système jusqu'au dernier état cohérent (sans erreur de causalité) avant ts(m) par le mécanisme de rollback.

Traitement à la réception d'un message négatif m^-

Lors de la réception d'un message négatif par un processus, deux cas peuvent se présenter: soit le message est dans le futur local, soit il est dans le passé local.

- A la réception de m^- dans le futur local, deux cas peuvent se présenter: soit son message positif m^+ est dans la liste des messages d'entrée non-traités (cas 5 de la figure 4.3) et ils s'annulent mutuellement dans *inputQueue*; soit son antimessage n'existe pas (cas 6 de la figure 4.3) et il est chronologiquement inséré dans la liste des messages d'entrée. Mais le message m^- ne sera jamais traité. Son rôle est alors d'empêcher un rollback à l'arrivée de m^+. Ainsi, les messages m^- sont mis en attente de leurs messages positifs m^+ pour qu'ils s'annulent mutuellement.

- A la réception de m^- dans le passé local, deux cas peuvent se présenter: soit son message positif m^+ est dans la liste des messages d'entrée (cas 7 de la figure 4.3) et il faut retourner l'état de l'ensemble du simulateur jusqu'au dernier état cohérent avant $ts(m)$ par le mécanisme de rollback; soit m^- est chronologiquement inséré dans *inputQueue* (cas 8 de la figure 4.3).

Remarque 7
Dans les cas 1, 3, 6, 8 de la figure 4.3, l'antimessages m^- devance son message positif m^+. Le PL n'entreprend aucun traitement à partir de m^- avant l'arrivée de son message positif m^+.

4.5 Contrôle global du simulateur

Le contrôle local dans le mécanisme de distorsion temporelle laisse certaines questions sans réponse. Comment peut-on s'assurer, malgré tous les rollbacks, que le simulateur avance globalement? Comment peut-on détecter la terminaison globale du simulateur? Comment peut-on gérer les erreurs et les Entrée/Sortie avec les rollbacks? Et comment peut-on éviter le manque de place mémoire pour sauver des états et des messages? Comment maîtrise-t-on le contrôle du flux des messages? Le contrôle global résout ces questions.

La stratégie optimiste nécessite de sauver les valeurs successives des variables locales de chaque processus et la liste de tous les messages envoyés et reçus. Pour pouvoir libérer la place en mémoire correspondant à une partie de ces sauvegardes et pour pouvoir effectuer les E/S, on introduit la notion d'**horloge virtuelle globale** ou GVT. Le concept de GVT est la base du contrôle global, une propriété instantanée globale du simulateur.

A un instant donné, le GVT représente la borne inférieure des temps virtuels des rollback possibles: toutes les actions effectuées à une date simulée antérieure à GVT ne seront jamais annulées. Elles sont appelées les *actions définitives ou sûres.* Seules les E/S antérieures au GVT sont réellement réalisées. Les états (ou messages) sauvés à une date inférieure au GVT peuvent être oubliés: ce sont *les états (ou messages) fossiles* et la récupération de leur espace mémoire s'appelle la *collection des fossiles.*

Comme il est impossible de déterminer la valeur exacte instantanée de GVT dans un environnement réparti, on se contente de son estimation par une borne inférieure. Elle se définit à un instant par la date minimale parmi la plus petite LVT de tous les PLs, la plus petite date d'expédition des messages envoyés mais qui ne sont pas encore reçus (*messages en transit*) et la plus petite date d'expédition des messages non-traités dans les PLs. Cette valeur estimée de GVT permet:

- la validation du passé de la simulation et la réalisation des opérations définitives appartenant à ce passé (par exemple les E/S et les calculs statistiques);
- la récupération de l'espace mémoire occupé par les états et les messages fossiles;
- la détection de la fin de la simulation.

Calcul de GVT

Un algorithme réparti et exécuté à intervalles réguliers permet de faire progresser l'estimation de GVT: $GVT_{estimé}$. Plusieurs algorithmes ont été proposés pour estimer GVT [Sam85], [Bel90], [LL90b], [CK91], [DFW94] et [XGUC95]. Nous développerons ultérieurement en détail ces algorithmes de GVT (cf. la troisième partie). Ils garantissent les deux propriétés suivantes [Lin94]:

- **Sûre**: la valeur de $GVT_{estimé}$ est garantie d'être inférieure ou égale à celle de *GVT réel*.
- **Précise**: la valeur de $GVT_{estimé}$ est garantie d'être plus grande que celle des $GVT_{estimé}$ du passé, c'est-à-dire qu'elle a progressé comme le fait le *GVT* réel [JS85].

Caractéristique du protocole de synchronisation

Dans la distorsion de temps, le *GVT* ne décroît pas et, de plus, il progresse si le modèle simulé est correct (c'est-à-dire s'il ne provoque pas d'impasse, ne produit pas une infinité de messages au même temps simulé, ne boucle pas dans le traitement d'un événement [JS85]). Elle est monotone non-décroissante.

Un processus optimiste ne se bloque que lorsqu'il n'a plus aucun événement dans son *inputQueue* et que tous les processus sont bloqués. Cela veut dire qu'il n'y a plus aucun événement à traiter dans le système et que la simulation est terminée.

4.6 Conclusion

Un *PL* optimiste traite les messages d'événement sans respecter la contrainte de causalité. Lorsqu'il viole la contrainte de causalité locale, une erreur de causalité survient; il la détecte et la corrige. La relation de causalité est donc respectée a posteriori.

Le protocole de synchronisation est très robuste et travaille avec plusieurs niveaux de circuits dans le réseau simulé. Plusieurs rollbacks peuvent être lancés simultanément sans aucune précaution spéciale de synchronisation. Contrairement au protocole de synchronisation pessimiste, il n'y a aucune possibilité d'impasse dans le protocole de synchronisation optimiste, tout simplement parce qu'il n'a aucun blocage ou attente dans le mécanisme de synchronisation. Mais les rollbacks en cascades peuvent mettre en péril l'efficacité du système. Nous verrons dans la partie suivante comment on peut rendre plus efficace l'exécution de la distorsion temporelle en réduisant le nombre de rollbacks.

Chapitre 5

Comparaison des approches de simulation

Cette section établit une comparaison entre la simulation sûre et la simulation optimiste.

5.1 Comparaison

La simulation sûre et la simulation optimiste partagent toutes deux les propriétés suivantes:

- le paradigme des processus logiques: le problème à simuler est d'abord décomposé en un ensemble de processus physiques PPs, puis, à chaque processus physique (PP), se voit associé un processus logique (PL).
- la communication entre les PLs par envoi de messages d'événements: chaque message d'événement représente une interaction entre les PPs et détermine à un instant particulier de la simulation. Les deux méthodes sont asynchrones; elles autorisent certains processus à être en avance ou en retard sur d'autres afin d'exploiter au maximum le parallélisme dans le système.

A partir de cette approche commune de modélisation, les méthodes sûre et risquée développent deux mécanismes distincts de synchronisation.

5.1.1 Différences dans le paradigme de simulation

Soit un PL reçevant une suite de messages estampillés $M1, M2, ..., avec Mi = (m_i, t_i)$.

Dans le mécanisme de synchronisation sûre, lorsqu'un PL reçoit un message d'événement $Mi = (m_i, t_i)$, il simule le comportement de son PP dans l'intervalle de temps compris entre t_{i-1} et t_i. Dans ce mécanisme, les messages sont reçus des canaux d'entrée dans l'ordre croissant de l'estampille et envoyés sur les canaux de sortie dans l'ordre croissant des estampilles afin d'éviter l'erreur de causalité.

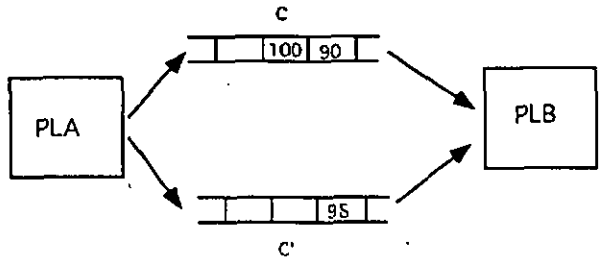


Figure 5.1: Canal c' utilisé rarement

Dans le mécanisme optimiste, lorsqu'un processus reçoit un message d'événement (m_i, t_i), celui-ci simule le comportement de son PP à l'instant t_i uniquement et non sur l'intervalle de temps compris entre t_{i-1} et t_i. Le PL envoie par la suite des messages dont l'estampille est supérieure ou égale à t_i, mais pas inférieure.

5.1.2 Restrictions statiques

Dans le mécanisme sûr, le modèle exécutable est représenté par un réseau de processus logiques (PL) reliés par des canaux orientés. Un PL peut avoir plusieurs canaux d'entrée ou de sortie mais la topologie du réseau est statique. Ceci n'est pas une contrainte facile à remplir car elle rend impossible la modification dynamique de la topologie du réseau au cours d'une exécution. De plus, le coût de ce mécanisme est proportionnel aux canaux rarement utilisés dans le réseau (cf. figure 5.1).

Dans le mécanisme optimiste, il n'y a pas de canaux qui relient les PLs entre eux. Un PL reçoit les messages qui lui sont envoyés dans une liste (file d'attente) de messages en entrée. La topologie d'interaction des PLs est entièrement dynamique.

Une autre différence entre les approches réside dans l'envoi et la réception des messages. Dans une approche sûre, un PL envoie et reçoit les messages dans l'ordre croissant des estampilles. En contre partie, dans un mécanisme optimiste, les messages sont envoyés et reçus dans un ordre quelconque, bien qu'une réception effectuée dans l'ordre croissant des estampilles contribue à l'efficacité du système.

Exemple 3 (Canal au flux irrégulier selon la figure 5.1)
Si le processus PL_A à l'instant $t = 80$ envoie un message $(m, 100)$ le long du canal c au PL_B, alors le PL_A ne peut plus envoyer de messages (m, t) avec $t < 100$. Il est fréquent qu'au cours de la simulation, le processus PL_A veuille envoyer plus tard à l'instant $t = 90$ un message $(m, 95)$ au processus PL_B. Ce message entraîne la préemption du message $(m, 100)$. Selon l'approche sûre, la préemption de messages se réalise en définissant un canal supplémentaire ici c'. Ainsi, le coût de résolution de l'impasse réside dans l'usage rare des canaux. Dans ces situations, le mécanisme optimiste se révèle particulièrement efficace.

5.1.3 Mécanismes de synchronisation

La simulation sûre et la simulation optimiste diffèrent fondamentalement par leur mécanisme de synchronisation [J+87]. Le premier est dit sûr dans le sens qu'aucun processus PL ne reçoit de message désynchronisé. En pratique, cette condition signifie qu'un PL demeure bloqué aussi longtemps qu'un de ses canaux d'entrée est vide. Un PL peut être frappé de famine si un de ces canaux d'entrée n'est pas utilisé au cours de la simulation

Si un canal d'entrée est rarement utilisé par un PL, ce dernier restera pratiquement bloqué au cours de la simulation.

Dans le second mécanisme de synchronisation où intervient la distorsion temporelle, le processus récepteur ne possède qu'une seule liste en entrée où les messages d'événement sont chronologiquement ordonnés par leur estampille. Le mécanisme de distorsion temporelle est optimiste (ou risqué) dans le sens qu'il autorise un PL à recevoir des messages désynchronisés, ce qui peut induire une violation de la contrainte de causalité (erreur de causalité) dans le simulateur. Dès cet instant, le PL rétablit la synchronisation en émettant des antimessages et en retournant au dernier état cohérent (qui aura dû être sauvegardé) sans erreur de causalité. L'état d'un PL doit être sauvegardé régulièrement, d'où le coût important de mémoire dans sa mise en œuvre.

Le désavantage du mécanisme sûr provient:

- de la stratégie de blocage d'un PL quand un ou plusieurs de ses canaux d'entrée sont vides, conduisant à un blocage et souvent à une impasse.
- de la présence de circuits dans le réseau d'interaction des processus conduisant à une impasse locale, pouvant s'étendre au réseau entier.

Le problème majeur de la réalisation d'un simulateur sûr est la résolution de l'impasse. Les différentes stratégies sûres se distinguent les unes des autres par la stratégie employée pour résoudre ce problème.

Dans la distorsion temporelle, l'évitement ou la résolution de l'impasse n'ont pas lieu d'être; car celle-ci dispose d'un mécanisme de contrôle local pour assurer la contrainte de causalité locale entre les processus et d'un mécanisme de contrôle global basé sur l'horloge virtuelle globale ou GVT ("Global Virtual Time") qui valide les actions de manière irréversible dans l'ensemble du système.

5.1.4 Étude de la performance: force et faiblesse

La mise en œuvre de la distorsion temporelle implique des coûts en mémoire considérables pour sauvegarder les états et contrôler les antimessages en vue d'exécuter correctement un rollback. Un rollback utilise des ressources supplémentaires alors que le coût d'une synchronisation sûre réside entièrement dans la technique utilisée pour gérer l'impasse.

La synchronisation risquée contracte ses coûts lorsque le modèle simulé est en activité. Par contre, la synchronisation sûre contracte ses coûts quand le modèle simulé est inactif. Les foyers susceptibles d'impasse sont situés autour des canaux d'entrée rarement utilisés (famines) et la présence de circuit dans le réseau simulé.

5.2 Conclusion

Les méthodes optimistes fondent leur espoir sur le mécanisme de rollback, malgré le coût de sauvegarde élevé; des succès significatifs ont été obtenus dans un grand nombre d'applications. Les méthodes sûres sont bien adaptées à certaines classes de problèmes où la topologie du réseau simulé est statique. Elles permettent d'intégrer plus facilement les connaissances spécifiques à l'application. Elles peuvent être réalisées sous forme d'une bibliothèque (package) dans laquelle le code de l'algorithme de synchronisation est optimisé. Par la suite, l'utilisateur n'a plus qu'à configurer les modules de la simulation dans son application spécifique.

L'approche sûre est supérieure à celle optimiste quand la simulation peut être décomposée en un réseau de *PLs* statiques dans lequel tous les canaux logiques ont un trafic régulier de messages [Fuj89a]. En revanche, dans un réseau dynamique où des paires de *PLs* interagissent rarement, l'approche optimiste est préférable car elle fournit de meilleures performances.

L'étude comparative des mesures de performance [Fuj90b] de la distorsion temporelle et du mécanisme de synchronisation sûre (évitement d'impasse et détection-guérison d'impasse) appliquée aux réseaux de file d'attente permet la conclusion suivante: la performance de la distorsion temporelle est comparable ou nettement supérieure à celle des algorithmes de synchronisation sûre. La distorsion temporelle exploite toute la puissance du parallélisme dans le modèle, sans recours à des informations spécifiques de l'application. Malgré ces résultats encourageants, les coûts en mémoire relatifs peuvent considérablement dégrader la performance du simulateur [FTG88a].

L'objectif des chapitres de la deuxième partie est la réduction des divers coûts qui pénalisent l'efficacité de la simulation optimiste. La mise en œuvre de la distorsion temporelle pour réduire le temps d'exécution et pour exploiter le parallélisme inhérent au système simulé pose les problèmes de rollbacks en cascades, d'optimisme excessif en simulant beaucoup d'événements incertains que nous tentons de résoudre de manière dynamique en combinant la théorie de contrôle et la simulation optimiste normale. En effet, il s'agit de décrire un schéma dynamique d'exécution dans un environnement réparti garantissant les trois propriétés suivantes: la *sûreté*, la *vivacité* et la *terminaison*. La sûreté du simulateur réparti exprime le fait que si le simulateur réparti se termine alors il produit le même résultat qu'une exécution séquentielle quand elle se termine. La vivacité exprime le fait que la plus petite estampille du message non traité est une fonction croissante au cours de la période de simulation. La terminaison exprime le fait que l'exécution répartie termine à un instant donné ou peut être terminée par un événement particulier.

Partie II

Etude du contrôle local de la simulation optimiste

Chapitre 6

Optimisation de la distorsion temporelle

Introduction

Jusqu'ici, tous les arguments du protocole d'exécution de la distorsion temporelle supposent qu'on dispose suffisamment de mémoire libre pour sauvegarder les états et les messages de chaque processus *(histoire du processus)* afin de traiter un éventuel rollback. En réalité, la mémoire n'est pas illimitée et toutes les stratégies d'optimisation de la distorsion temporelle se ramènent au problème de la mémoire. Il arrive qu'on termine de manière anticipée la simulation par manque d'espace mémoire. Une limitation de mémoire pénalise l'efficacité de la simulation. Il en découle que la performance d'exécution dépend fortement de la place mémoire à disposition [LP91]. La gestion de mémoire dans la distorsion temporelle poursuit deux buts: rendre le protocole opérationnel sur une architecture multiprocesseurs, voire à "mémoire limitée" et rendre efficace l'exécution de la distorsion temporelle en présence de place mémoire suffisante.

Les différentes approches d'optimisation proposées pour maîtriser la gestion de mémoire peuvent être classées en cinq catégories:

- les mécanismes de rollback pour rétablir la synchronisation;
- la technique de récupération de mémoire;
- la sauvegarde des états;
- les protocoles utilisés pour limiter l'optimisme (le nombre d'événements qui ne sont pas sûrs);
- les protocoles pour réduire le nombre de rollbacks ou *stabiliser le système.*

Ce chapitre est exclusivement consacré aux différentes approches mises en œuvre pour rendre efficace le mécanisme de contrôle local de la simulation optimiste. Les cinq sections suivantes présentent les cinq catégories ci-dessus. La dernière section fournit une synthèse des travaux existants et définit notre axe de recherche du mécanisme de contrôle adaptatif.

6.1 Différents mécanismes de rollback

Dans la distorsion temporelle, une erreur de causalité survient lorsqu'un processus reçoit un message désynchronisé. La procédure de rollback rétablit la synchronisation perdue suite à une erreur de causalité. L'optimisation de ces mécanismes réside dans la réparation des dommages causés pour rétablir la synchronisation perdue. Les différentes méthodes d'optimisation proposées à partir de l'annulation agressive ("aggressive cancellation") sont: l'annulation paresseuse ("lazy cancellation") [Gaf88], [Wes88]; la réévaluation paresseuse ("lazy reevaluation"); la relaxation des rollbacks ("rollback relaxation") [WP94]; la chasse aux calculs erronés ("wolf calls") [MWM88] et l'annulation directe.

L'objectif des diverses méthodes de rollbacks consiste à réduire les causes de calculs prévisionnels inutiles afin d'accroître l'efficacité de la simulation.

6.1.1 Annulation agressive

Dans la version originale de distorsion temporelle [Jef85], lorsqu'un processus reçoit un message désynchronisé, il enclenche immédiatement le mécanisme de rollback et d'envoi d'antimessages. Cette méthode d'annulation est appelée l'*annulation agressive*.

6.1.2 Annulation paresseuse

Avec l'annulation paresseuse, pour améliorer la performance de l'annulation agressive [LSW89], les processus n'envoient pas immédiatement les antimessages à la réception du message désynchronisé (m, t). Ils attendent de voir si le traitement de la "resimulation" du message désynchronisé engendre les *mêmes messages*. Si, à la fin de l'exécution, les messages engendrés n'affectent pas les messages déjà traités, aucun antimessage ne sera émis. Sinon on envoie les antimessages de la liste des messages de sortie *outputQueue*.

L'annulation paresseuse évite les rollbacks inutiles mais elle nécessite de la mémoire supplémentaire pour mémoriser la "resimulation" et un temps d'attente pour annuler les états courants resimulés.

Selon l'application, l'annulation paresseuse améliore ou détériore la performance du simulateur [JR91], [Fer95].

6.1.3 Réévaluation paresseuse

L'optimisation de la réévaluation paresseuse est similaire à l'annulation paresseuse. Elle filtre les antimessages à envoyer quand une erreur de causalité survient. Elle gère à la place d'un état, des vecteurs d'états [Wes88] pour faire la différence des états. Un vecteur d'états est l'ensemble des états entre deux sauvegardes consécutifs. Considérons le cas où l'état du processus est le même après l'exécution de l'événement désynchronisé: si aucun nouveau message n'interfère, il est évident que la réexécution des événements du rollback serait identique à l'exécution originale; par contre si certains états ont changé, on n'a pas besoin d'envoyer tous les antimessages. On peut sauter les états non modifiés. Cette

technique nécessite la comparaison des vecteurs d'état pour déterminer si l'état a changé ou non.

6.1.4 Méthodes de prévision

Ces méthodes tentent d'exploiter (implicitement ou explicitement) les propriétés prévisionnelles ("look ahead") afin d'optimiser la simulation. L'aspect fondamental du calcul consiste à exploiter l'invariance du comportement de la méthode d'annulation paresseuse et celle de réévaluation paresseuse à l'arrivée du message désynchronisé, ce qui ressemble étroitement à la propriété de prévision dans les approches sûres. L'annulation paresseuse bénéficie de l'avantage de prévision, même si l'application n'était pas explicitement programmée à cet effet. La programmation explicite de prévision dans l'application a aussi ses avantages [BCLU89] mais il est difficile de les programmer ou de les intégrer explicitement dans les approches optimistes.

6.1.5 Chasse aux calculs erronés ("Wolf calls")

C'est une technique selon laquelle un message désynchronisé ("traggler message") provoque l'envoi de messages de contrôle spéciaux pour arrêter rapidement la propagation des calculs erronés. Les processus atteints par les erreurs de calculs sont notés quand une erreur est détectée. Ce schéma exige qu'on connaisse d'une part la vitesse en temps-réel de la propagation des erreurs de calcul et, d'autre part, le temps nécessaire pour transmettre les messages de calculs. La détermination des valeurs de ces quantités peut être très difficile dans certains systèmes.

Le désavantage de cette approche est que certains calculs corrects peuvent être inutilement gelés. De plus, certaines optimisations efficaces telles que l'annulation paresseuse ("lazy cancellation") ne peuvent être intégrées au simulateur.

6.1.6 Annulation directe

Dans la distorsion de temps, il est important qu'on puisse arrêter les calculs incorrects rapidement pour qu'ils ne se propagent pas trop loin dans le simulateur. Sinon, les antimessages essayeront toujours de rattraper les messages positifs de calculs erronés [Jef85]. Une manière de prévenir ce comportement est de rendre les antimessages prioritaires sur les messages positifs; ou, chaque fois qu'un événement $e1$ ordonnance un autre $e2$, un pointeur est laissé sur $e1$ vers $e2$. Ce pointeur sert à annuler les actes de $e2$ dès que $e1$ doit être annulé. Ce schéma offre deux avantages [LMBS83], [LL90a]: il réduit les coûts d'annulation de messages et capture rapidement les calculs erronés pour minimiser les dommages qu'ils causent.

6.1.7 Relaxation des rollbacks

La relaxation du rollback [WP94] résout le problème de l'espace mémoire en divisant l'ensemble des processus (PLs) de la simulation en deux sous-ensembles disjoints: l'ensemble des processus ayant besoin d'espace mémoire et l'ensemble de ceux qui n'en ont pas besoin

pour s'exécuter. Les processus du premier ensemble utilisent les états internes et sauvegardés pour le traitement d'un événement, alors que ceux du second ensemble n'utilisent jamais les états internes, ni la liste des états sauvés pour traiter ses événements.

Le premier ensemble de processus traite ses éléments de la partition de manière classique. Les processus du second ensemble sont qualifiés par la "relaxation du rollback". Dans la relaxation du rollback, aucun état des processus du second ensemble n'est sauvé durant la simulation. A la place, le processus reconstruit son dernier état cohérent à la réception du message désynchronisé à partir des messages de son *inputQueue*.

6.2 Techniques de récupération de mémoire

Dans la distorsion temporelle, chaque processus gère les listes suivantes:

- *unprocessedIQ*, la liste des messages non-traités (futurs) en entrée;
- *processedIQ*, la liste des messages déjà traités en entrée;
- *stateQueue* qui contient la liste des états récemment sauvés ou histoire du processus;
- *outputQueue*, la liste des antimessages en sortie.

Remarque 8
La réunion des deux premières listes correspond à la liste des messages d'entrée *inputQueue*; ainsi pour un PL donné, on a:

$unprocessedIQ \cup processedIQ = inputQueue$ et $unprocessedIQ \cap processedIQ = \emptyset$.

Les trois dernières listes sont gérées de manière similaire. Un mécanisme de collection de fossile libère la place mémoire des messages ou des états ayant une estampille inférieure au GVT. Le mécanisme dépend fortement de la fréquence d'estimation de GVT. D'où, une fois encore, l'intérêt d'un algorithme efficace de détermination de GVT.

La gestion de la place mémoire de la première liste est liée au problème de contrôle de flux propre aux systèmes répartis. Pour un débit élevé, la liste des messages futurs (non-traités) peut déborder. Que fait-on alors à la réception d'un nouveau message?

Les différentes approches proposées pour récupérer la place mémoire de la liste des messages futurs en entrée sont:

- **Le renvoi de messages à l'expéditeur** ("sendback") [Jef85]: si un message (m, t) ne peut être traité par manque de place mémoire, le message est retourné à son expéditeur. L'expéditeur retourne son état à l'état avant l'expédition du message retourné. Le retour d'un message est un mécanisme analogue au rollback du temps virtuel.
- **L'annulation en arrière** ("cancelback") [Gaf88]: si un message estampillé par t ne peut être traité par manque d'espace mémoire, on essaye de récupérer la place occupée par des messages ayant une estampille supérieure à t en envoyant leurs antimessages. Le but est de récupérer les espaces occupés par les messages et les états

les plus *loin tous dans le futur* et de les utiliser pour des traitements plus urgents des messages *dans le présent.*

- Le **retour en arrière artificiel** ("artificial rollback") [Lin92] consiste à exécuter des rollbacks sans avoir reçu de message désynchronisé. Chaque fois que GVT est déterminé, chaque processus exécute la collection des fossiles et un rollback artificiel.

Remarque 9
Une simulation répartie est dite optimale du point de vue de l'occupation mémoire si elle est capable d'accomplir une simulation en utilisant la même quantité de mémoire que la simulation séquentielle du même modèle [Jef90]. Lin et Preiss [LP91] ont étudié de façon formelle les besoins en espace mémoire de la simulation répartie et concluent que l'approche sûre [CM81] et la distorsion temporelle munie de l'algorithme de gestion mémoire de Gafni [Gaf88] ne sont pas optimum. En revanche, le retour en arrière artificiel ("artificial rollback") ou l'annulation en arrière ("cancelback") dans une distorsion temporelle sont optimum du point de vue l'occupation mémoire.

6.3 Sauvegarde des états

Les diverses solutions pour réduire le coût de sauvegarde d'états peuvent se résumer en trois catégories:

1. la conception de *carte accélératrice* spécifique une d'application donnée qui est une solution matérielle coûteuse [FW95];

2. la sauvegarde des états seuls modifiés par rapport à l'état précédent ou la *sauvegarde incrémentale des états* [Bau93];

3. la sauvegarde d'états à des intervalles de temps réguliers ou irréguliers appelée *sauvegarde périodique des états* [Bel92]; c'est la plus prometteuse de toutes ces approches.

En général, les états sont sauvés après l'exécution de chaque événement. Mais par manque de mémoire disponible, Lin [LL89] propose de sauver périodiquement les états aprés n exécutions (l'intervalle de sauvegarde): c'est la sauvegarde périodique des états. La réception d'un message désynchronisé peut entraîner une erreur de causalité. Elle se corrige en exécutant un rollback qui comporte deux phases:

P1 un rollback à l'état cohérent précédent;

P2 la reconstitution ("coast-forward") de l'état cohérent juste avant le message désynchronisé où les messages d'événement ne sont pas envoyés aux PLs du système.

L'inconvénient de la sauvegarde périodique des états est qu'une grande période produit une grande phase de reconstitution de l'état cohérent.

L'analyse de la sauvegarde périodique des états [LL90a], [PW93b] permet de conclure:

- qu'elle est une méthode efficace d'optimisation du temps d'exécution et de la place mémoire consommée;
- qu'elle pourrait cependant être améliorée en rendant dynamique la période de sauvegarde à la place d'une période statique.

6.4 Protocoles pour limiter l'optimisme

Ces protocoles tentent de réduire le nombre d'*événements non sûrs ou incertains* dans chaque processus et dans le simulateur entier. Le but recherché par ces protocoles est d'avancer les LVTs de telle manière que leurs valeurs soient toujours à peu près égales. Ainsi, la longueur des rollbacks est courte et le nombre d'événements incertains (risqués) est réduit.

6.4.1 Méthode de la fenêtre temporelle

En général dans une simulation, les PLs ont entre eux une différence considérable de temps d'exécution et d'activité d'événement. Ainsi, il y a dans le système des PLs lents et rapides. Ce déséquilibre fait que la valeur de LVT des PLs rapides est en avance sur celle des PLs lents. Ce déséquilibre entraîne une augmentation du nombre de rollbacks dans le système et peut mener à la terminaison de la simulation par manque de place mémoire. Pour prévenir ce comportement, l'amplitude des LVTs entre le PL le plus lent et le PL le plus rapide est confinée dans un intervalle de temps ou ***fenêtre temporelle***. Le but de la fenêtre temporelle est de permettre à tous les PLs de faire progresser leurs horloges locales de manière équilibrée.

La méthode de la fenêtre temporelle est utilisée pour éviter de propager trop loin les calculs incorrects dans le futur simulé. Elle permet de limiter considérablement l'intervalle pour traiter des événements incertains (les risques ou optimismes) [SBW88]. On reproche à cette méthode de ne pas distinguer un bon calcul prévisionnel d'un mauvais, ce qui pourrait empêcher la progression d'un bon calcul. De plus, la détermination de la taille de fenêtre n'est pas définie.

Les paramètres considérés dans la détermination de la taille de la fenêtre temporelle sont la vitesse la fréquence du calcul de GVT, la granularité des événements traités et le comportement du rollback dans le système. Ainsi, la détermination de la taille est une décision complexe et difficile à prendre.

L'étude de la technique de la fenêtre temporelle [PW93a], [MT93], [RWJ89] permet de déduire qu'on obtiendra une optimisation efficace en faisant varier dynamiquement la taille des fenêtres temporelles, liées à chaque processus.

6.4.2 Protocole d'ordonnancement

En général, le nombre de PLs nécessaires dans une simulation n'est pas égal au nombre de processeurs disponibles. Particulièrement lors d'une simulation complexe, il est probable que le nombre de *PLs* soit supérieur au nombre de processeurs. Ainsi, chaque processeur se charge de l'exécution de plusieurs *PLs*. Lorsqu'un événement fait intervenir plusieurs *PLs* prêts à s'exécuter, un algorithme d'ordonnancement doit désigner le prochain *PLs* à s'exécuter.

Une manière d'équilibrer les LVTs des PLs sur un processeur dans la distorsion temporelle est de définir un algorithme d'ordonnancement spécifique au simulateur. Les diverses stratégies d'ordonnancement [BM90], [LL91a] des PLs sur un processeur se résument à donner la priorité soit au processus ayant le plus petit LVT, soit au processus ayant la plus petite estampille des messages dans sa liste en entrée, soit selon la stratégie d'ordonnancement à tour de rôle. Aucune de ces stratégies ne tient compte du comportement des rollbacks pour réduire le temps d'exécution et la place mémoire utilisée durant l'exécution. Elles ne tiennent non plus pas compte de la qualité de synchronisation. Ces stratégies sont développées en détail plus tard. Toutes ces stratégies, en voulant optimiser localement un aspect, détériorent l'efficacité globale du simulateur.

6.5 Protocole de stabilisation

Cette section passe en revue les différents travaux visant à réduire le nombre de rollbacks dans la distorsion temporelle: c'est-à-dire à ***stabiliser la simulation***. Nous étudions les diverses techniques proposées pour réduire le nombre de rollbacks.

La distorsion temporelle limitée ("bounded Time Warp") [TX92] et la **réduction de la fenêtre temporelle** ("window based throttling") [RWJ89], [Bri90] restreignent les risques de traitement en permettant à chaque processus de s'exécuter dans une fenêtre temporelle bornée. La taille de la fenêtre est statistique et la même pour tous les processus de la simulation. Ces techniques restreignent les calculs prévisionnels indésirables. Elles limitent aussi l'optimisme des PLs effectuant des calculs prévisionnels corrects. Cette optimisation est difficile à utiliser car les différents modèles de simulation d'un domaine d'application nécessitent des fenêtres de tailles différentes pour une performance optimale. En outre, une taille de fenêtre statique inadaptée peut faire diminuer dramatiquement la performance du système.

La distorsion temporelle adaptative ("adaptive Time Warp") [BH90] introduit la synchronisation sûre dans le système en admettant qu'un processus dont le traitement mène à un nombre élevé de rollbacks attend durant une période *BW* ("bounded window ou window size"). La période *BW* varie en fonction du temps pour établir le dernier état correct ("duration of wasted lookahead computation"). Mais déterminer une fonction critique de *BW* qui correspond à une application de simulation est un problème complexe. A moins qu'une bonne fonction (courbe) ne soit obtenue, la période (taille) de *BW* obtenue maximisera l'efficacité du système. L'hypothèse que la période *BW* soit la même pour tous les processus invalide le modèle car tous les processus n'ont pas la même vitesse d'exécution. Par conséquent, la valeur initiale de *BW* serait incorrect. Un autre

désavantage de cette méthode est que les processus se bloquent avant le traitement de chaque événement ou après le traitement d'un nombre donné d'événements.

La méthode d'ordonnancement basée sur les anti-messages ("penalty based Throttling") [RWJ89] est un algorithme d'ordonnancement qui ordonnance les PLs qui reçoivent le moins d'antimessages. Les PLs sont pénalisés en fonction du nombre d'antimessages reçus. Les PLs qui ont un nombre élevé d'antimessages sont ignorés jusqu'à ce qu'ils aient réduit leur nombre d'anti-messages à zéro. L'inconvénient majeur de cette technique est que le processeur n'est pas utilisé quand tous les PLs du système sont pénalisés, ce qui entraîne des attentes actives du cycle CPU ("busy wait"). Les performances de cette technique sont décourageantes.

Une combinaison de l'ordonnancement d'après la plus petite estampille et la répartition équilibrée des charges ("load balancing") pour réduire le nombre de rollback a été proposée par Glazer et Tropper [GT93]. L'équilibrage des charges est réalisé en modifiant le découpage du temps "time slice" des PLs en fonction du paramètre suivant "le taux de progression de la simulation" du PL. Cette approche est similaire à d'autres approches [MT93], [PW93a] basées sur le découpage du temps ("time slice") où la différence réside dans la manière de modifier le paramètre en fonction du découpage du temps. Les résultats de cette technique ne sont pas testés dans un environnement réparti mais dans un environnement multiprocesseurs simulé, sans prendre en compte les facteurs du coût de l'ordonnanceur du système d'exploitation et du coût de la charge du système induite par d'autres processus utilisateurs.

La méthode de fenêtre bornée [Bry88] ("adaptive Time Ceiling") est basée sur un concept de modification de la taille de fenêtre, bien que les tailles de la fenêtre soient choisies dans un ensemble de valeurs discrètes. Le paramètre de contrôle est très simple et ne fait pas intervenir d'information globale telle que le nombre d'antimessages ou le nombre de messages positifs.

Les approches hybrides ("hybrid approaches") ont été développées en combinant les algorithmes risqués et sûrs. SRADS, avec des rollbacks locaux [DR90] et des calculs spéculatifs risqués [Meh91], traite localement des événements incertains, bien que les résultats ne soient pas émis à d'autres PLs sur un autre processeur du simulateur. Ainsi les rollbacks sont confinés dans les PLs d'un processeur; de telles approches sont dites **agressives sans risque**. **La méthode de saut avec répit** ("Breathing Time Buckets") [Ste93] est similaire au protocole de SRADS avec des rollbacks locaux, mais elle a l'avantage de modifier dynamiquement la taille des fenêtres temporelles d'après le concept d'horizon global et local relatif aux événements. Mais cette technique fournit une faible performance lorsque le système d'application possède une faible capacité de calcul de prédiction. Un autre inconvénient de cette technique est qu'elle doit d'abord exécuter tous les messages avant de déterminer l'horizon global des événements pour s'assurer qu'aucun message n'est en transit.

La distorsion temporelle avec répit ("Breathing Time Warp") [Ste93] est une combinaison de la méthode de saut avec répit ("Breathing Time Buckets") et de la distorsion temporelle ("Time Warp"). Cette technique partitionne l'ensemble des événements en deux sous-ensembles: E_{proche} pour ceux proches de GVT et E_{autres} pour les autres. Ensuite, elle suppose que les événements proches de GVT ont une faible probabilité de

subir un rollback. Les événements de E_{proche} sont exécutés selon le mécanisme de distorsion temporelle et les autres événements de E_{autres} sont exécutés par la suite en utilisant la technique de saut avec répit ("Breathing Time Buckets"). En outre, le mécanisme ne fournit aucun critère pour déterminer dynamiquement les deux sous-ensembles en vue de minimiser le temps d'exécution.

La méthode de la restriction de l'espace de la recherche ("Bounded Lag") [Lub89] utilise la distance minimum entre deux PLs comme pour définir les événements sûrs. Selon cette technique, le programmeur doit fournir à priori la borne inférieure entre deux messages désynchronisés. **La méthode de rollbacks sélectionnés** ("Filtered Rollback") [LSW89] assouplit la fenêtre temporelle sûre pour permettre la violation de la contrainte de causalité et rétablir un état cohérent en utilisant les techniques de rollback optimiste. Mais cette technique ne fournit aucun moyen de déterminer les distances entre les PLs et de les modifier dynamiquement afin de rendre efficace leur exécution.

D'après les différents travaux présentés pour réduire le nombre de rollbacks et accroître l'efficacité du simulateur réparti optimiste, on peut déduire que les méthodes combinant l'approche sûre et optimiste nécessitent des informations explicites pour obtenir une exécution efficace.

6.6 Synthèse des travaux précédents

La distorsion temporelle est un protocole de synchronisation optimiste. Sa réalisation pour réduire le temps d'exécution et pour exploiter le parallélisme inhérent au système simulé pose les problèmes:

- de rollback en cascade (ou en chaîne) qui annulent les actions simulées, ce qui mène à l'instabilité du système;
- de sauvegarde excessive d'états et du risque de simuler un nombre démesuré d'événements "incertains" de manière anticipée;
- de l'inefficacité des algorithmes d'ordonnancement des PLs sur chaque processeur.

Une synthèse des différents travaux visant à résoudre ces problèmes et à rendre efficace la simulation répartie peut se résumer à:

1. La réduction du nombre de rollbacks en minimisant les calculs prévisionnels (qui ne sont pas définitifs) inutiles. Pour cela, la distorsion temporelle utilise les mécanismes de rollback et la fenêtre temporelle.
2. La récupération de la place mémoire utilisée au cours de la simulation. Pour cela, la distorsion temporelle utilise les techniques de récupération de mémoire et les méthodes de sauvegarde des états.
3. L'apport d'un algorithme d'ordonnancement des processus sur un processeur. La distorsion temporelle utilise les différentes stratégies d'ordonnancement des processus logiques sur un processeur.

4. L'apport d'une méthode hybride (sûre et optimiste) nécessitant des informations explicites du modèle pour obtenir une exécution efficace.

6.7 Orientation de la recherche

D'aprés les différents travaux, une meilleure efficacité du simulateur réparti optimiste proviendrait de la modification de la méthode de la distorsion temporelle:

- la méthode de contrôle combinant la simulation risquée et la simulation sûre doit être dynamique au cours de la simulation;
- l'approche de la fenêtre temporelle pour limiter les risques (l'optimisme) est prometteuse; mais il reste à définir un mécanisme pour initialiser et modifier dynamiquement la taille des fenêtres au cours de l'exécution;
- l'approche de la sauvegarde périodique est aussi prometteuse; mais il reste à définir et à initialiser un mécanisme qui en modifie adaptativement la période de sauvegarde d'états (ou le nombre d'événements traités entre deux calculs consécutifs de GVT) au cours de la simulation;
- un algorithme efficace d'ordonnancement, prometteur dans la distorsion temporelle, découlerait d'une stratégie dynamique d'ordonnancement des PLs au cours de l'exécution en tenant compte de l'avancement de leurs LVTs.

Plusieurs modifications sont intégrées au mécanisme original de la distorsion temporelle pour stabiliser et améliorer la synchronisation. Chaque processus logique dispose d'une fonction dynamique et évolutive afin de contrôler le risque encouru (nombre d'événements incertains traités).

Notre tâche consiste d'abord (au chapitre suivant) à identifier les indices de performance de chaque PL en vue d'établir sa fonction d'évaluation appelée sa **qualité de synchronisation**. Cette fonction représente sa fonction économique. Ensuite, nous étudierons dans le dernier chapitre de cette partie (chapitre 8) la mise en œuvre du contrôle adaptatif (dynamique) de chaque PL qui apporte une solution aux problèmes posés:

- en modifiant adaptativement l'intervalle de sauvegarde d'états (le nombre d'événements traités en deux calculs de GVT consécutifs) en fonction de la "qualité de synchronisation";
- en limitant le risque ou les événements incertains simulés dans une fenêtre temporelle en fonction de la qualité de synchronisation;
- en favorisant l'ordonnancement des PLs dont la "qualité de synchronisation" est la meilleure.

Chapitre 7

Un Modèle de Contrôle dynamique optimiste

Résumé

La distorsion temporelle ("Time Warp") est un protocole de synchronisation de la simulation optimiste. Sa réalisation pour réduire le temps d'exécution et pour exploiter le parallélisme inhérent au système simulé pose les problèmes suivants:

- rollbacks en cascade (ou en chaîne) qui annulent les actions simulées pour rendre le *système instable*;
- sauvegarde excessive d'états;
- optimisme excessif prenant le risque de simuler un nombre démesuré d'*événements incertains* de manière anticipée;
- inefficacité des algorithmes d'ordonnancement des *PLs* sur chaque processeur.

En général, ces problèmes sont résolus individuellement de manière statique. Ces diverses tentatives n'ont qu'un succès limité, à cause de l'affectation statique des paramètres de contrôle intervenant dans la résolution du problème. Dans ce chapitre nous abordons les problèmes par la **simulation dynamique optimiste** où chaque *PL* dispose d'une fonction dynamique.

Ce chapitre s'inspire de la théorie du contrôle des systèmes adaptatifs pour définir le modèle de contrôle dynamique d'un processus logique (*PL*) de la simulation répartie. L'intégration du contrôle adaptatif nécessite quelques modifications du mécanisme classique de la distorsion temporelle pour stabiliser et améliorer la synchronisation. Chaque *PL* dispose d'une fonction d'évaluation, appelée **qualité de la synchronisation**. Elle est établie à partir des indices de performances de la simulation optimiste.

7.1 Introduction

La *distorsion du temps* ("time Warp") est un mécanisme de synchronisation pour la communication asynchrone des processus logiques *PL*s. Chaque *PL* a sa date virtuelle locale ou *LVT* ("Local Virtual Time") et s'exécute indépendamment des autres de manière asynchrone jusqu'à ce qu'il reçoive un *message désynchronisé*. Dès cet instant, le *PL* rétablit le dernier état cohérent qu'il a sauvegardé pour rétablir la synchronisation. Ensuite *PL* envoie des anti-messages aux *PL*s voisins pour annuler les actions postérieures à l'estampille du message désynchronisé. Nous supposons disposer d'un algorithme d'approximation de GVT ("Global Virtual Time") ou du moins d'un algorithme équivalent à celui de *GVT*. Le mécanisme de synchronisation dans la distorsion temporelle ne peut avoir de succès que si le coût du rollback est minimisé.

Nous proposons d'évaluer chaque *PL* à l'aide de ses paramètres de contrôle pour suivre dynamiquement la qualité de ses solutions partielles. La qualité d'une solution dépend du coût du risque encouru et du travail utile dans un intervalle de temps. Cette information est utilisée pour contrôler l'exécution de chaque processus afin d'améliorer l'efficacité de la simulation.

Nous proposons un nouveau concept basé sur la *qualité de synchronisation* pour faire tendre dynamiquement les paramètres de sortie de chaque *PL* vers une meilleure qualité de synchronisation à chaque estimation de *GVT*. La qualité de synchronisation d'un *PL* dépend du risque courant encouru pour accomplir la synchronisation à chaque estimation de *GVT*. La technique de contrôle adaptatif est appliquée localement à chaque processus pour maximiser l'efficacité du système entier. En particulier, le contrôle adaptatif d'un processus est conçu en s'inspirant du contrôle d'un modèle de référence d'un système ouvert adaptatif [Kok91]. Les indices de performance (*IP*) permettent de contrôler individuellement chaque processus pour limiter ses risques et, par conséquent, maximiser le débit de ses activités de synchronisation. Ainsi, un nouveau paradigme de synchronisation auto-adaptative d'un processus est établi.

Ce chapitre comporte cinq sections. La section 2 présente les objectifs de notre approche; la section 3 présente la théorie de contrôle en mettant l'accent sur le contrôle adaptatif ; les sections 4 présente le modèle contrôle adaptatif d'un *PL* à partir de ses indices de performance. Ensuite, nous établissons la fonction d'évaluation selon la "qualité de synchronisation" d'un *PL*.

La dernière section conclut en présentant les avantages du concept dynamique par rapport à l'approche classique.

7.2 Objectif de notre approche

La simulation répartie est conçue à la base pour accélérer la simulation d'un programme de simulation par événements discrets dans un environnement réparti. Dans ce chapitre, notre objectif est d'intégrer un mécanisme de contrôle dynamique à l'exécution de la simulation optimiste pour rendre efficace le système simulé. L'intégration du contrôle dynamique (adaptatif) dans le contrôle local de la distorsion temporelle suscite les questions suivantes:

- doit-on contrôler les risques encourus sur des informations locales ou globales, ou sur une combinaison des deux?
- comment peut-on contrôler les activités anticipées de manière à limiter les risques encourus?
- comment évaluer et apprécier adaptativement la qualité d'une solution partielle de chaque *PL*?
- comment définir la meilleure fonction de synchronisation ou la "fonction optimale" d'un processus à partir de son état courant?
- quels sont les paramètres de contrôle ou les indices de performance qui déterminent la fonction économique de synchronisation?
- comment établir cette fonction sans geler les *PL*s du système au cours de la détermination de *GVT* ou son équivalent *LTmin GVT* local?

En effet, le but de la "fonction économique de synchronisation" est de tendre dynamiquement l'exécution de la simulation optimiste vers l'exécution de la simulation sûre. Par opposition à la simulation optimiste simple, notre approche contrôle et exploite les événements incertains. Cette approche est basée sur la mesure des indices de performance de chaque processus qui reflètent l'évaluation de sa synchronisation. La solution proposée ajuste localement les paramètres de contrôle de la fonction de synchronisation de chaque processus, compte tenu de sa performance locale, pour obtenir la performance globale du simulateur. En fait, le problème posé sous cet angle est un *problème de décision markovien* consistant à trouver une stratégie optimale de contrôle qui maximise la fonction de synchronisation [Kok91]. La solution proposée est une intégration des techniques de contrôle adaptatif dans le modèle de la simulation répartie. Nous donnons donc un aperçu des techniques de contrôle de système avant d'aborder l'étude des fondements de la fonction économique de synchronisation qui contrôle dynamiquement un processus.

7.3 Théorie du contrôle

La théorie du contrôle étudie la modification du comportement d'un système dynamique pour qu'il réalise des buts désirés [Kok91]. Ces buts servent à maintenir les paramètres de sortie constants lorsqu'ils sont connus à l'avance ou à assurer que l'ensemble du système se comporte conformément aux critères d'optimisation spécifiés. Les phases déterminantes dans le contrôle d'un système dynamique sont:

1. la modélisation mathématique du système;
2. l'identification des indices de performances du système à partir des données expérimentales: la phase d'*identification du système*;
3. le traitement des paramètres de sortie et leur usage pour déduire les paramètres de contrôle en entrée qui réalisent le comportement optimal.

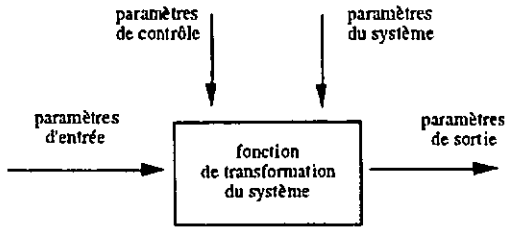


Figure 7.1: Contrôle de système ouvert

Cette section présente brièvement le contrôle des systèmes ouverts, systèmes fermés et systémes adaptatifs (évolutifs).

7.3.1 Contrôle de système ouvert

Un système ouvert simple est composé d'un ensemble de paramètres d'entrée, d'une fonction de transformation et d'un ensemble de paramètres de sortie selon la figure 7.1.

La fonction de transformation détermine le comportement du système. Elle dépend des paramètres du système, des paramètres de contrôle et des paramètres d'entrée. La fonction de transformation ne tient pas compte de l'état précédent du système. Ainsi, une modification adaptative de l'un des paramètres aboutit à des paramètres de sortie incorrects ou non désirés. Par conséquent, seul un environnement stable est adapté au contrôle d'un système ouvert.

7.3.2 Contrôle de système fermé

Un contrôle de système fermé tente de corriger les problèmes de contrôle des systèmes ouverts, en autorisant la rétroaction ("feedback") des paramètres de sortie selon la figure 7.2. La fonction e désigne la différence entre les paramètres d'entrée et l'élément de feedback provenant de la sortie. La fonction *e* sert à régulariser les objectifs du systéme. La fonction de transformation dépend de l'histoire du système: de l'état passé et de l'état courant du système. Les changements des paramètres de contrôle et ceux du modèle peuvent modifier les paramètres de sortie. Bien que le contrôle de système fermé s'adapte aux environnements variables, il devient inutilisable pour contrôler des systèmes robustes où le temps de réponse est l'une des caractéristiques essentielles des opérations du système. Car il ne tient pas compte des paramètres de contrôle évolutifs dans le système. Ainsi, pour obtenir un système robuste (stable), nous avons besoin de modifier continuellement divers paramètres de contrôle et du modèle; d'où l'intérêt du contrôle des systèmes adaptatifs.

7.3.3 Systèmes adaptatifs

Le terme **adaptation** se définit par *l'aptitude d'un individu à modifier sa structure ou son comportement pour répondre harmonieusement à des situations nouvelles de son environnement* [PETIT ROBERT]. Un système inconnu, aussi complexe soit-il, est simplement un système rétroactif non-linéaire [Kok91] qu'on pourrait simuler et contrôler à l'aide d'une boucle de contrôle adaptatif. L'étude de contrôle des systèmes adapatatifs portent sur

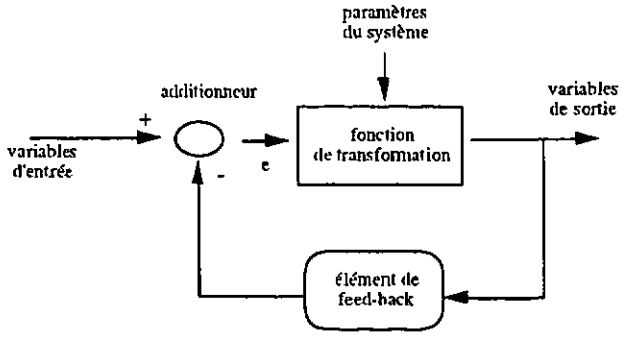


Figure 7.2: Contrôle de système fermé

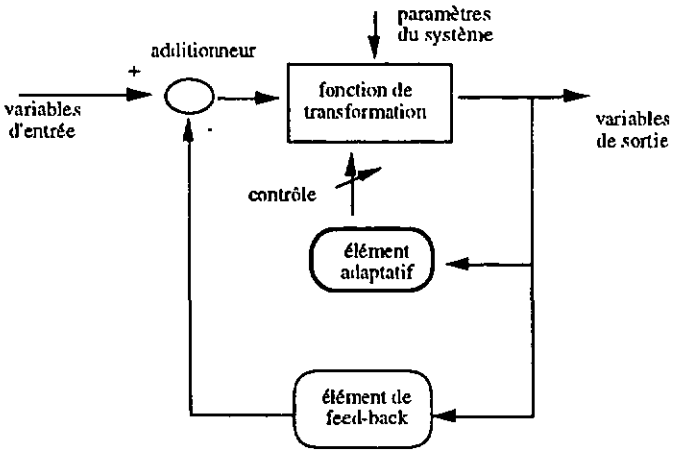


Figure 7.3: Contrôle de système auto-adaptatif

deux catégories de systèmes: les *systèmes auto-adaptatifs* ("self-tuning regulator adaptive systems") et *les systèmes adaptatifs sur un modèle de référence* ("model reference adaptive systems") [Kok91].

Systèmes auto-adaptatifs

La figure 7.3 montre le contrôle d'un système auto-adaptatif où les paramètres de contrôle du système sont variables.

Le mécanisme de rétroaction ("feed-back") est la procédure de contrôle adaptatif. Il sert, à chaque instant, à modifier les paramètres de contrôle pour obtenir les critères de la meilleure solution (solution optimale). La définition de la fonction *élément adaptatif* dépend de:

1. l'estimation des paramètres du modèle;

2. l'estimation des paramètres de contrôle pour fournir la meilleure solution partielle.

Le contrôle des systèmes adaptatifs fournit différentes solutions potentielles à un problème; il reste à retenir la (es) meilleure(s). La fonction de transformation du système

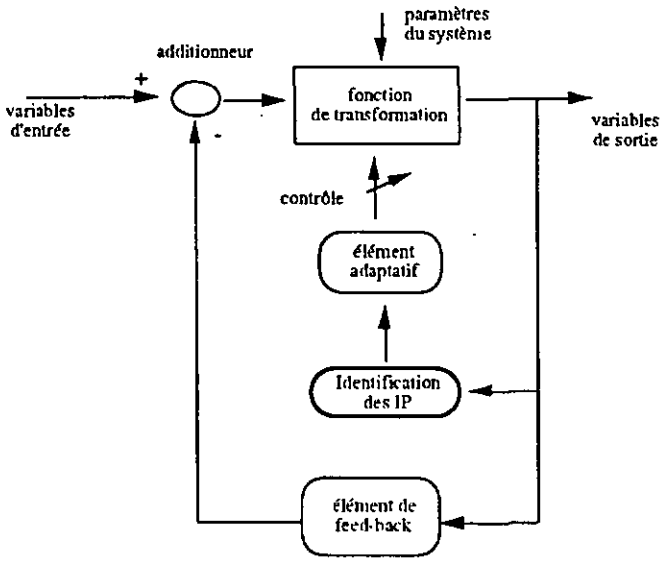


Figure 7.4: Contrôle d'un modèle fermé de SAMR

contrôlé dépend de son environnement évolutif, du modèle temporel, des paramètres d'entrée, de la modification des paramètres de contrôle ou des changements de la performance des objectifs.

Le contrôle auto-adaptatif est difficile à concevoir pour les raisons suivantes:

- la plupart des systèmes ont des fonctions de transformation pratiquement impossibles à déterminer;
- l'estimation des paramètres est difficile, voire impossible.

Pour surmonter ces problèmes, le contrôle de systèmes adaptatifs sur un modèle de référence (SAMR) a été proposé [Kok91].

Contrôle de systèmes adaptatifs sur un modèle (SAMR)

Le contrôle de SAMR construit d'abord un modèle référentiel du système adaptatif (problème d'identification du système), puis établit la stratégie de contrôle sur le modèle référentiel du système adaptatif. Les figures 7.4 et 7.5 présentent respectivement le contrôle adaptatif d'un modèle référentiel de système fermé et le contrôle adaptatif du modèle référentiel de système ouvert. On identifie dans ces modèles les *indices de performance IP* qui indiquent la qualité (performance) instantanée du système. La procédure de contrôle adaptatif, appelé *élément adaptatif*, est conçue pour optimiser les paramètres de sortie en ajustant automatiquement les paramètres de contrôle. La structure principale du système se présente sous la forme d'une boucle fermée ou d'une boucle ouverte. L'élément adaptatif représente la procédure de contrôle adaptatif.

La procédure de contrôle adaptatif fonctionne en 3 phases appelées:

1. **Identification** qui définit la phase de la caractérisation du système où la valeur IP (Indices de Performance) est mesurée;

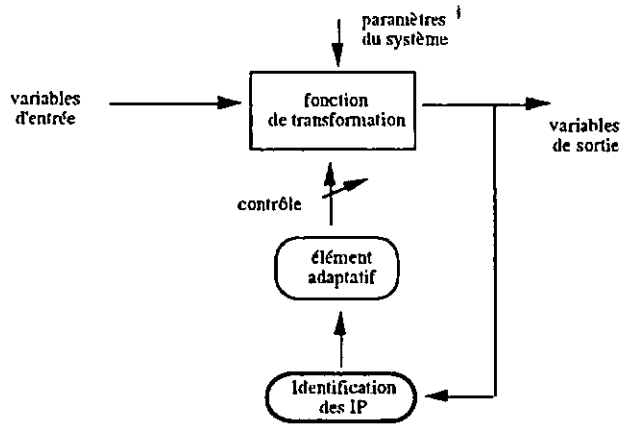


Figure 7.5: Contrôle d'un modèle ouvert de SAMR

2. **Décision** qui est la phase à laquelle les mesures des IP sont utilisées pour décider de la manière dont la performance du système se rattache à la solution optimale désirée;

3. **Modification** qui définit la phase où les paramètres (de contrôle, du modèle et du système) se modifient en vue de tendre vers la solution optimale souhaitée, conformément aux phases ci-dessus d'identification et de décision.

7.4 Fondement du contrôle dynamique d'un PL

Nous considérons la simulation optimiste comme un ensemble de système de PLs selon la figure 7.6, interconnectés en fonction de la structure de communication du modèle simulé. D'après la théorie de contrôle des systèmes adaptatifs [Kok91], pour rendre le simulateur dans son ensemble presque optimal, il suffit d'optimiser l'exécution de chaque PL en contrôlant de manière adaptative ses paramètres de contrôle.

Quels sont les paramètres d'un PL adaptatif?

1. *Les variables d'entrée* d'un PL sont:
   (i) les événements dans le passé ou présent local;
   (ii) les événements dans le futur local;
   (iii) les antimessages reçus et
   (iv) la réception de messages de contrôle pour déterminer GVT.
2. *Les variables de sortie* d'un PL sont:
   (i) les messages positifs envoyés;
   (ii) les messages négatifs envoyés;
   (iii) le nombre de rollbacks et
   (iv) l'expédition de messages de contrôle pour déterminer GVT.

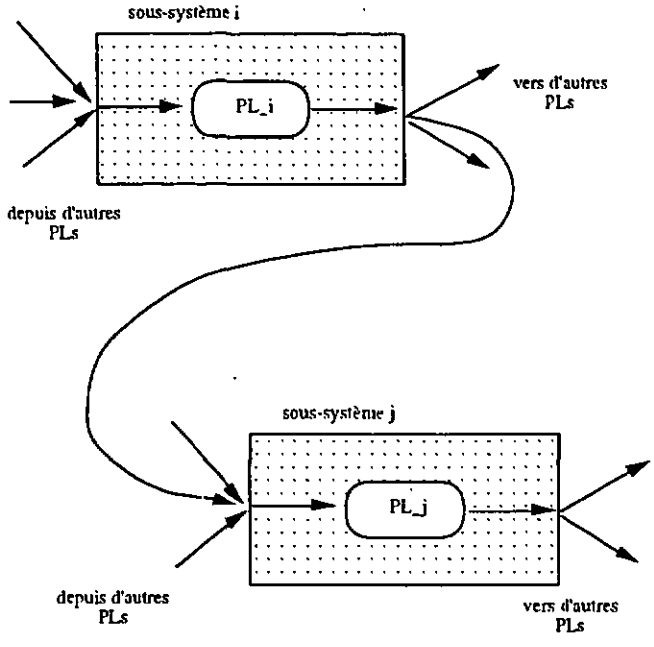


Figure 7.6: Système de PLs

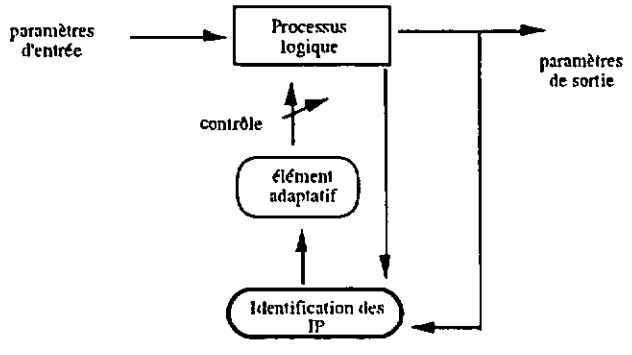


Figure 7.7: Modèle de contrôle adaptatif d'un PL

3. *Les paramètres de contrôle* d'un PL sont typiquement:

   (i) la limite des risques (optimisme);

   (ii) la période de sauvegarde d'états;

   (iii) la priorité d'ordonnancement des PLs sur un processeur selon les buts.

Les constantes du système n'influencent pas son évolution adaptative; par contre, on a besoin d'un mécanisme adaptatif (d'après la figure 7.7) qui modifie continuellement les paramètres de contrôle au cours de la simulation pour obtenir une meilleure solution.

En effet, il n'existe aucun moyen pour déterminer la fonction de transformation d'un système. Seul l'indice de performance IP indique l'avancement du système. Bien entendu, la phase difficile de la conception du contrôle adaptatif d'un PL réside dans la détermination des IP. Le paragraphe suivant présente la manière de choisir les IP adéquats caractérisant la performance d'un PL.

7.4.1 Concept de qualité de synchronisation

Idéalement, la simulation dynamique optimiste est optimale quand tous les PLs s'exécutent sans rollback. Cependant, en pratique, les PLs exécutent des rollbacks à chaque perte de synchronisation. Pour rendre le simulateur optimiste efficace, c'est-à-dire minimiser son temps d'exécution, l'environnement réparti devrait remplir les conditions suivantes:

- minimiser le nombre de rollbacks par PL;
- minimiser le nombre d'anti-messages envoyés par un PL;
- réduire le nombre d'événements à annuler et maximiser ainsi les calculs prévisionnels;
- maximiser le nombre d'événements sûrs (définitifs) dans un intervalle de temps donné, pour accroître le débit du système;

Toute action posée par un PL pour rendre son exécution efficace doit contribuer à améliorer sa "qualité de synchronisation"; sinon elle diminue la performance du système. Les paramètres de contrôle adaptatif de la fonction "qualité de synchronisation" $Q_{synchro}$, pour déterminer son meilleur état courant (le moins incertain par rapport aux états précédents), doivent être une combinaison des conditions ci-dessus. La fonction de contrôle doit refléter l'évolution du PL, ainsi que le nombre d'événements définitifs traités par seconde (c'est-à-dire le débit du PL).

Sous la forme d'une équation, nous définissons la fonction "qualité de synchronisation" $Q_{synchro}$ d'un PL comme le rapport du nombre E_d d'événements définitifs à chaque cycle de GVT par le temps δ_{cpu} écoulé pour les traiter:

$$Q_{synchro} = \frac{E_d}{\delta_{cpu}} \tag{7.1}$$

Cette approche simple pour déterminer $Q_{synchro}$ ne reflète pas le débit du traitement pour les raisons suivantes:

1. il n'est pas nécessaire que tous les événements aient le même temps d'exécution;
2. la fréquence des événements peut être variable; plus d'un événement peuvent survenir au même instant LV; ceci implique qu'il existe des instants auxquels plusieurs événements surviennent et d'autres où il y en a moins.

Cependant, si tous les événements ont un même temps d'exécution et surviennent à une fréquence constante, alors la formule 7.1 détermine la "qualité de synchronisation" de la simulation par événements discrets dans un environnement réparti. Le paragraphe suivant présente une définition et une formule qui tiennent compte des divers coûts intervenant dans un cycle de GVT.

7.4.2 Indice de performance:"qualité de synchronisation"

Supposons que la "qualité de synchronisation" $Q_{synchro}$ soit calculée à chaque cycle δ_{cpu} du calcul de GVT et que toutes les mesures considérées se rapportent à cet intervalle de temps. Si δ_{cpu} est petit, on passera plus de temps à évaluer $Q_{synchro}$ qu'à s'occuper de la simulation du modèle.

Rappelons qu'un cycle désigne l'intervalle de temps entre deux calculs de GVT.

Soient: E_t: le nombre total d'événements traités dans un cycle;

E_d: le nombre d'événements définitifs traités dans un cycle;

E_a: le nombre d'événements annulés dans un cycle;

M_n: le nombre d'anti-messages (ou négatifs) envoyés dans un cycle;

M_p: le nombre de messages positifs envoyés dans un cycle et

α: le rapport du nombre d'événements définitifs par le nombre d'événements traités.

$$\alpha = \frac{E_d}{E_t} \tag{7.2}$$

Idéalement, $\alpha = 1$ (et $E_a = 0$).

L_r: longueur moyenne des rollbacks par cycle;

N_r: nombre de rollbacks par cycle;

Propositions

La qualité de synchronisation d'un PL se base sur les propositions suivantes:

1. $\alpha \in [0..1]$, plus α est grand, plus la qualité de synchronisation est grande:

   $Q_{synchro} \approx \alpha$ où $\approx$ désigne une relation de proportionnalité;

2. chaque antimessage envoyé contribue à dégrader la performance de la qualité de synchronisation:

   $Q_{synchro} \approx 1/M_n$

3. chaque rollback correspond à une contre-performance d'établissement d'un état stable; l'ampleur de la désynchronisation est déterminée par la longueur moyenne des rollbacks dans un cycle:

   $Q_{synchro} \approx 1/(L_r \cdot N_r)$

7.4.3 Une approche de Modèle optimal

Pour que le système d'PLs soit efficace, il suffit que chaque processus logique (PL) s'exécute en un minimum de temps. A partir des propositions ci-dessus, on minimise le coût de la synchronisation en définissant la fonction de la qualité de synchronisation par:

$$Q_{synchro} = k \cdot \frac{\alpha}{L_r \cdot N_r \cdot M_n} = k \cdot \frac{E_d}{E_t \cdot L_r \cdot N_r \cdot M_n} \tag{7.3}$$

où k désigne une constante de proportionnalité.

Ainsi, la fonction $Q^i_{synchro}$ évalue le travail productif accompli par le PL_i à chaque cycle de GVT, en tenant compte des coûts contractés et de l'avancement de la simulation.

7.5 Conclusion

Le paradigme de la simulation dymanique optimiste repose sur le principe suivant: chaque PL dispose de ses paramètres de contrôle (nous en avons identifié ici trois) où chaque paramètre de contrôle est conçu pour optimiser un aspect particulier dans l'activité d'un PL. Les activitées d'optimisation des paramètres de contrôle sont rassemblées (réunies) dans l'unique fonction appelée **qualité de synchronisation**. Cette fonction est conçue pour accroître, en un cycle de GVT, le nombre des événements définitifs et réduire la période des calculs de prédiction de chaque PL, de manière à rendre sûr l'exécution d'un PL. Ainsi, on réduit dans tout le simulateur le nombre de rollbacks exécutés. Par conséquent, la simulation dynamique optimiste possède plus d'avantages que la simulation optimiste classique. Le prochain chapitre présente les avantages à utiliser la fonction de la qualité de synchronisation d'un PL pour optimiser les paramètres de contrôle de la simulation répartie optimiste.

Chapitre 8

Mise en œuvre du contrôle dynamique d'un PL

Résumé

Dans le chapitre précédent, nous avons défini le modèle de contrôle adaptatif d'un PL de la simulation répartie. Le contrôle adaptatif se base sur le concept de la *qualité de synchronisation.*

Ce chapitre présente la manière dont la qualité de synchronisation intervient dans le contrôle dynamique d'un PL pour apporter une solution aux problèmes classiques de la simulation optimiste:

1. en modifiant adaptativement la période de sauvegarde des états (le nombre d'événements traités en deux calculs de GVT consécutifs) au cours de l'exécution, en vue de faire régulièrement la collection de fossiles;

2. en modifiant dynamiquement la taille de la fenêtre temporelle d'un PL au cours de l'exécution afin de limiter la simulation des événements incertains;

3. en modifiant dynamiquement la priorité d'ordonnancement des PLs, en vue de favoriser les PLs dont la "qualité de synchronisation" est la meilleure.

Nous présentons les contributions de la simulation dynamique optimiste en utilisant le contrôle adaptatif dans chaque PL.

8.1 Introduction

Ce chapitre présente les avantages d'un contrôle adaptatif pour ajuster, à chaque cycle de GVT, les paramètres de contrôle d'un PL de la simulation répartie.

Il présente la manière dont la qualité de synchronisation intervient dans le contrôle dynamique d'un PL pour apporter une solution aux problèmes classiques de la simulation optimiste:

1. en modifiant adaptativement la période de sauvegarde des états (le nombre d'événements traités entre deux calculs consécutifs de GVT) au cours de l'exécution, en vue de faire régulièrement la collection de fossiles;

2. en modifiant dynamiquement la taille de la fenêtre temporelle d'un PL au cours de l'exécution afin de limiter le nombre d'événements incertains simulés;

3. en modifiant dynamiquement la priorité d'ordonnancement des PLs, en vue de favoriser les PLs dont la "qualité de synchronisation" est la meilleure.

Ce chapitre est structuré en trois sections. La première section se rapporte à l'étude de la période de sauvegarde des états au cours de l'exécution, la deuxième à la taille des fenêtres temporelles et la troisième à l'ordonnancement des PLs en présence de ressources (ici processeurs) limitées. Chaque section présente brièvement les travaux précédents dans son domaine, déduit la limite de ces travaux avant de proposer un modèle empirique qui se base sur la qualité d'un PL à chaque cycle. La dernière présente en conclusion la contribution apportée par la simulation dynamique optimiste.

8.2 Sauvegarde des états

Afin de permettre le rollback, chaque PL doit sauvegarder régulièment une partie ou l'ensemble de ses états: ceci correspond à la **sauvegarde d'états**. La sauvegarde d'états a deux effets négatifs sur la simulation. D'une part, la consommation excessive de mémoire peut entraîner la rupture anticipée de la simulation par faute de mémoire. D'autre part, le temps d'exécution nécessaire à sauver les états et à restituer les états sauvés pénalise l'efficacité de la simulation. Il y a plusieurs manières d'aborder le coût d'exécution et le coût d'occupation de la mémoire dans une simulation optimiste.

Rappelons que les diverses solutions pour réduire le coût de sauvegarde d'états peuvent se résumer en trois catégories:

- la conception d'une carte accélératrice spécifique à une application donnée qui est une solution matérielle coûteuse [FTG92];

- la seule sauvegarde des changements d'un état à un autre dans le système: la *sauvegarde incrémentale des états* [Bau93]; et

- la sauvegarde d'états à des intervalles de temps réguliers ou irréguliers appelée *sauvegarde périodique des états* [Bel92].

Les deux dernières solutions logicielles sont les plus appliquées.

8.2.1 Sauvegarde incrémentale d'états

Une manière de diminuer le coût des rollbacks est la sauvegarde incrémentale d'états. Contrairement à la sauvegarde périodique d'états qui sauve le vecteur d'états entier d'un PL, la sauvegarde incrémentale sauve uniquement les changements par rapport au vecteur d'états précédent [PW93b]. Elle sauve uniquement des modifications partielles d'un vecteur d'états à un autre. La restitution d'état est accomplie en appliquant à chaque instant la fonction inverse du changement partiel incrémental des vecteur d'états afin de rétablir un vecteur d'état complet à cet instant. Bien que cette technique réduise le temps de sauvegarde, elle augmente le temps de restitution des états sauvés selon la longueur du rollback.

Dans les applications où l'état d'un PL varie beaucoup, la sauvegarde incrémentale est moins prometteuse que la sauvegarde périodique des états [CGU+94].

8.2.2 Sauvegarde périodique d'états

Une autre manière de réduire le coût de la sauvegarde des états est de réduire la fréquence des sauvegardes. Dans le cas le plus simple, un PL sauve son état après le traitement de chaque message d'entrée. Dans le cas général, un PL sauve ses états après le traitement de χ messages d'entrée, où $\chi > 1$. La période de sauvegarde d'états ("checkpoint interval") χ est le nombre de messages d'entrée traités entre deux sauvegardes. Le cas simple correspond à $\chi = 1$.

Si un rollback survient quand $\chi = 1$, alors l'état précédent cohérent peut être mémorisé directement dans la liste des états sauvés. Cette liste fournit directement les états possibles qui permettent à un PL de reprendre son exécution, lors de la réception d'un message désynchronisé. Par contre, si un rollback survient lorsque $\chi > 1$, l'état du point de reprise n'est pas directement fourni. Il est obtenu à partir de l'état précédent cohérent sauvé (dans la liste des états sauvés) en resimulant des messages d'entrée jusqu'à l'état cohérent de reprise (cf. figure 8.1). Remarquons qu'une des pénalités associée à $\chi > 1$ est le temps nécessaire à reconstruire les états non-sauvegardés. La manière de reconstruire l'état non-sauvegardé lorsqu'on en a besoin est une considération importante dans la réalisation de distorsion temporelle. Bien que l'état soit facilement reconstruit en resimulant les messages d'entrée, les messages de sortie engendrés pendant la reconstruction de l'état ne peuvent pas être envoyés. Le temps de reconstruction dépend (i) du temps nécessaire pour qu'un événement soit resimulé, (ii) du nombre d'événements à resimuler et (iii) de la probabilité de devoir reconstruire un état.

Temps de traitement d'une sauvegarde

Le temps de la sauvegarde périodique est le temps nécessaire pour accomplir une opération de sauvegarde. Ce temps comporte deux parties : le temps nécessaire pour allouer la place mémoire à un état et le temps nécessaire pour copier l'état courant dans la mémoire réservée. Ce temps est directement associé au temps d'activation de sauvegarde des états d'un PL.

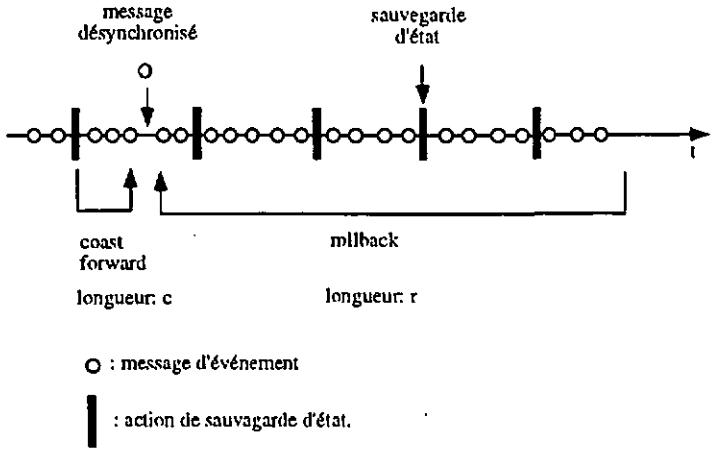


Figure 8.1: Restitution des états sauvegardés pour $\chi = 4$

Coûts de la date virtuelle globale (GVT)

Aucun PL ne peut ramener sa date en dessous de GVT. Les états sauvegardés, dont l'estampille est inférieure au GVT (les fossiles), ne sont plus utiles. La récupération de la mémoire allouée aux fossiles est appelée la ***collection de fossiles***. Le calcul de GVT introduit deux types de pénalité temporelle: le temps pour estimer GVT et le temps pour exécuter l'algorithme de la collection des fossiles.

8.2.3 Optimisation de la période de sauvegarde

La sauvegarde d'états ne survenant pas après le traitement de chaque événement, un rollback doit parfois réexécuter plusieurs événements intermédiaires avant le traitement de l'événement désynchronisé (cf. figure 8.1). Cette réexécution est appelée la phase de *reconstruction correcte* ("coast forward phase").

La période optimale de sauvegarde doit être un équilibre entre le temps passé à sauver les états et celui passé à exécuter la phase de la reconstruction correcte (cf. figure 8.1) d'un PL. Plus précisement:

- en augmentant la fréquence des sauvegardes d'états, on réduit le temps consommé par la routine de sauvegarde d'états et l'espace mémoire consommée par la copie des états.

- en augmentant la période de sauvegarde d'états, on accroît le nombre moyen d'événements durant la phase de la reconstruction correcte; ceci provient du fait que la longueur moyenne de la reconstruction correcte c est proportionnelle à la période de sauvegarde d'états.

Dans les applications de la simulation répartie, la ressource temps d'exécution et plus importante que la ressource mémoire. Par conséquent, il faut se contenter de minimiser le temps d'exécution pour rendre le simulateur réparti efficace.

Analyse des méthodes de la période de sauvegarde

L'étude de la sauvegarde des états [LL90a] permet de tirer les conclusions suivantes:

- La sauvegarde périodique des états est une méthode d'optimisation très efficace car elle fournit une optimisation temporelle (temps d'exécution) et spatiale (la place mémoire).
- La sauvegarde périodique des états peut être un moyen pour déterminer l'optimum de la période de sauvegarde au cours de la simulation.

8.2.4 Algorithmes adaptatifs de la sauvegarde d'états

Les modèles analytiques sont élaborés d'après les hypothèses suivantes: (i) le traitement d'un rollback a lieu dès que l'exécution courante est terminée; ainsi il n'y a pas de préemption de message d'événement; (ii) le processus sauve son état après χ événements exécutés (χ désigne la période de sauvegarde des états); (iii) les rollbacks sont indépendants les uns des autres, c'est-à-dire que les longueurs de rollbacks sont indépendantes les unes des autres.

Différents modèles analytiques

Il y a plusieurs approches pour ajuster la période de sauvegarde au cours de la simulation. Trois techniques particulières pour l'ajustement dynamique de la période de sauvegarde d'états sont examinées dans la littérature: le modèle de Lin [LPLL93]; la dérivation de Palaniswamy [PW93b] et l'approche Rønngren [RA94]. Chacune de ces méthodes dérive de l'étude analytique (théorique) d'un modèle de simulateur réparti optimiste où elle utilise le modèle pour dériver une formule en vue d'ajuster dynamiquement la valeur de période de la sauvegarde des états. Malheureusement, les méthodes analytiques aboutissent à une formulation assez complexe difficile à évaluer et à contrôler au cours de l'éxécution. Ainsi, elles s'avèrent toutes inapplicables. C'est pourquoi nous présentons une méthode empirique qui tienne compte de la qualité de synchronisation d'un PL pour ajuster dynamiquement la période de sauvegarde au cours de l'exécution.

8.2.5 Empirique pour la période de sauvegarde d'états

Par la nature dynamique d'un PL, il est difficile d'attribuer une constante à la période de sauvegarde d'états pour obtenir une performance optimale. Une manière raisonnable d'adapter la période de sauvegarde est de l'ajuster dynamiquement pour chaque PL au cours de la simulation, compte tenu de la fréquence de ses rollbacks, de la taille de sa liste des états sauvés et de la granularité de ses événements à exécuter. Cette période doit être influencée par la qualité courante instantanée de la synchronisation. L'heuristique est basée sur la notion de "qualité de synchronisation" $Q_{synchro}$ d'un PL présentée au chapitre précédent.

Le simulateur initialise la période de sauvegarde d'états χ à une valeur aléatoire à l'aide d'un générateur aléatoire. Ensuite, à chaque nouvelle valeur de GVT (car à cet

instant on sait exactement le nombre d'événements définitifs), on détermine une nouvelle période de sauvegarde d'états selon la qualité de synchronisation instantanée $Q_{synchro}$. Si la qualité de synchronisation $Q_{synchro}$ croît, alors la période de sauvegarde χ croît d'une unité. Alors que lorsque $Q_{synchro}$ décroît, la période de sauvegarde décroît d'une unité et si $Q_{synchro}$ est constant, la période de sauvegarde demeure inchangée.

L'algorithme réalisant l'heuristique est présenté dans la figure 8.2.

$Q_{synchro}$:qualité de synchronisation
periode: la période de sauvegarde d'états
k: constante d'ajustement dynamique de la période (=1)
begin
Initialisation:
periode = valeur initiale de la période;
$Q^{old}_{synchro}$ = valeur initiale;
loop
attendre une nouvelle estimation de GVT;
déterminer la qualité de synchronisation par:
$Q^{new}_{synchro} := k \cdot \frac{E_d}{E_f \cdot L_r \cdot N_r \cdot M_n}$
if $Q^{new}_{synchro} > Q^{old}_{synchro}$ then
$periode := periode + k$;
elsif $Q^{new}_{synchro} < Q^{old}_{synchro}$ then
$periode := periode - -k$;
endif
$Q^{old}_{synchro} := Q^{new}_{synchro}$
end loop
end.

Figure 8.2: *Algorithme de contrôle adaptatif de la période de sauvegarde d'états*

8.3 Taille adaptative de la fenêtre temporelle

Dans un simulateur optimiste utilisant le protocole de distorsion temporelle, on peut avoir des processus logiques lents et rapides. Si un PL lent défait continuellement les activités d'un PL rapide, le coût du rollback croît et la performance du simulateur entier se dégrade. Afin de diminuer le nombre de rollback provenant d'un tel comportement du simulateur, il est nécessaire de ralentir les PLs rapides dans l'espoir d'équilibrer leur flux de communication avec des PLs lents (ou inversement). Pour limiter l'avancement des processus logiques rapides sur les lents, Sokol et ses collègues [SBW88] ont proposé la notion de la *fenêtre temporelle* (cf. la figure 8.3) qui autorise de simuler uniquement les événements dont l'estampille appartient à un intervalle de temps ou fenêtre temporelle. L'apport de ce protocole réside dans le fait qu'il permet de simuler des événements avec une faible variation d'estampille. L'amplitude de la variation est au plus égale à la taille de la fenêtre temporelle.

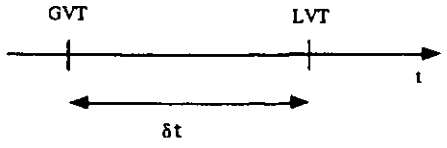


Figure 8.3: Fenêtre temporelle δt

Cette section présente l'impact de l'ajustement dynamique de la taille des fenêtres temporelles dans une synchronisation de distorsion temporelle.

Le paragraphe suivant présente les travaux relatifs aux tailles statiques des fenêtres temporelles. Le paragraphe 8.3.2 présente l'algorithme adaptatif de la taille des fenêtres temporelles pour équilibrer la progression des PLs du simulateur.

8.3.1 Taille statique des fenêtres temporelles

Chaque PL exécute les événements de ses calculs de prédiction dans une fenêtre temporelle $[GVT, GVT + \delta t]$ où δt représente la taille de la fenêtre (cf. la figure 8.3). Le risque de chaque PL est contenu dans un intervalle de temps. Ainsi, deux événements $e1$ et $e2$ peuvent être simulés en parallèle si et seulement si $|ts(e1) - ts(e2)| < \delta t$. Lorsque LVT d'un PL avance jusqu'à la borne supérieure de la fenêtre temporelle, il attend que la valeur de GVT avance pour continuer son activité [SS90].

La difficulté majeure de cette approche réside dans l'attribution d'une valeur à la taille de la fenêtre temporelle δt. Car pour une grande valeur, elle rejoint exactement le mécanisme de synchronisation de la distorsion du temps, alors qu'une petite valeur fait perdre les avantages des calculs de prédiction corrects.

Une valeur statique de la taille de la fenêtre temporelle peut contraindre inutilement un PL dont les calculs de prédiction sont corrects et ne pas contraindre d'autres PLs qui effectuent des calculs incorrects. Par conséquent, nous proposons une extension de la fenêtre temporelle en intégrant dans chaque PL un mécanisme d'ajustement dynamique de la taille de sa fenêtre temporelle δt en fonction de sa qualité de synchronisation. Les PLs, dont la qualité de synchronisation $Q_{synchro}$ est grande, font des calculs corrects prévisionnels. Alors ils auront une grande taille de fenêtre temporelle. Alors que ceux dont la valeur de $Q_{synchro}$ décroît font des calculs prévisionnels incertains et sont contraints à une petite taille de fenêtre.

8.3.2 Ajustement dynamique de la fenêtre temporelle

Cette section présente un algorithme empirique pour modifier dynamiquement la taille de la fenêtre temporelle de chaque PL, au cours de la simulation, en utilisant sa qualité de synchronisation. L'heuristique de l'algorithme est basée sur l'observation suivante:

1. Les PLs qui disposent d'un nombre élevé de rollbacks doivent être ralentis. Ils représentent les PLs rapides dans le simulateur. Ils doivent avoir une petite taille de fenêtre temporelle pour ne pas devoir exécuter des rollbacks de grande ampleur. La taille de leur fenêtre est diminuée d'une constante.

2. Les PLs avec un petit nombre de rollbacks peuvent augmenter la taille de leur fenêtre temporelle selon la valeur de leur qualité de synchronisation $Q_{synchro}$ à chaque cycle de GVT. Si la valeur de $Q_{synchro}$ croît, la taille de la fenêtre temporelle augmente, sinon elle diminue.

3. Les PLs dont le nombre de rollback est égal à zéro (qui n'ont jamais exécuté de rollback) peuvent avoir une grande taille de fenêtre. Ils peuvent augmenter leur taille de fenêtre d'une valeur constante (une unité) à chaque cycle de GVT. C'est le cas typique d'un PL source.

La structure algorithmique qui réalise l'heuristique de la fenêtre temporelle est représentée dans la figure 8.4.

Chaque PL est initialisé au plus petit commun multiple ($ppcm$) des temps de calcul de prédiction dans le simulateur. La qualité de synchronisation $Q_{synchro}$ est déterminée après chaque calcul de GVT. C'est à ce moment-là qu'on connaît le nombre d'événements définitifs au cycle. Lorsque la valeur de $Q_{synchro}$ décroît, la taille de la fenêtre temporelle décroît aussi.

En somme, pour limiter les risques, chaque PL utilise à chaque cycle de GVT un algorithme auto-adaptatif pour ajuster la taille de la fenêtre temporelle. Cette adaptation dépend de la valeur de sa qualité de synchronisation.

$Q_{synchro}$: qualité de synchronisation.
tailleFT: la taille de la fenêtre temporelle
k: constante d'ajustement de la taille des fenêtres (=1)
begin
  Initialisation:
    tailleFT = valeur initiale de la tailleFT
    $Q^{old}_{synchro}$ = valeur initiale;
  while not (endofSimulation) do
    attendre une nouvelle estimation de GVT
    déterminer la qualité de synchronisation par:
    $Q^{new}_{synchro} := k \cdot \frac{E_d}{E_t \cdot L_r \cdot N_r \cdot M_n}$
    if $Q^{new}_{synchro} \geq Q^{old}_{synchro}$ Then
      tailleFT := tailleFT + k;
    else
      tailleFT := tailleFT − k;
    endif
    $Q^{old}_{synchro} \leftarrow Q^{new}_{synchro}$
  end while
end.

Figure 8.4: *Algorithme de contrôle adaptatif de la taille de fenêtre temporelle*

8.4 Ordonnancement des *PLs*

En général, le nombre de *PLs* nécessaires dans une simulation n'est pas égal au nombre de processeurs disponibles. Particulièrement lors d'une simulation complexe où il est probable que le nombre de *PLs* soit supérieur au nombre de processeurs. Ainsi, chaque processeur se charge de l'exécution de plusieurs *PLs*. Lorsqu'un événement fait intervenir plusieurs *PLs* prêts à s'exécuter, un algorithme d'ordonnancement doit désigner le prochain *PLs* à s'exécuter.

Cette section traite exclusivement des algorithmes d'ordonnancement dans un simulateur optimiste utilisant la distorsion temporelle. Elle est organisée en trois paragraphes. Le premier présente les critères d'un "bon" algorithme d'ordonnancement; le second présente les différentes stratégies d'ordonnancement des *PLs* sur un processeur dans un simulateur optimiste; le troisième présente la stratégie d'ordonnancement basée sur le paradigme de la "qualité de synchronisation" $Q_{synchro}$.

8.4.1 Critères d'un algorithme d'ordonnancement

Les exigences d'un ordonnanceur sont multiples et complexes. Néanmoins, le but d'un algorithme d'ordonnancement est de toujours sélectionner le *PL* à s'exécuter de manière à ce que le temps d'éxécution total du simulateur soit minimal.

Un bon ordonnanceur de *PLs* sur un processeur dans un environnement réparti doit satisfaire les critères ci-dessous:

1. l'ordonnanceur devrait être simple et ne doit pas introduire des effets de bord dans le phénomène simulé;

2. chaque PL devrait être ordonnancé de manière à réduire le temps total d'exécution;

3. tous les *PLs* devraient avoir une chance équitable de s'exécuter; autrement (i) une impasse pourrait survenir à cause de la famine ou (ii) une instabilité à cause du nombre démesuré des rollbacks. Or, la présence d'une famine dans le simulateur résulterait d'une occupation excessive de la mémoire provenant soit d'un nombre excessif de messages non traités, soit d'un nombre excessif d'états sauvegardés suivis de la collection rare de fossiles.

8.4.2 Algorithmes d'ordonnancement de *PLs*

Nous examinons les trois algorithmes d'ordonnancement, [LL91a] utilisés en simulation optimiste. Ils sont tous non-préemptifs dans le sens où une fois qu'un *PL* commence le traitement d'un message d'entrée, il finit son traitement avant que l'ordonnanceur donne la main à un autre *PL*. De plus, l'arrivée d'un antimessage dans un *PL* n'interrompt pas son cycle courant du traitement d'événement.

1. **Ordonnancement Round-Robin (*RR*)**

Cet algorithme ordonnance les messages en entrée des *PL*s prêts (de la manière Round-Robin). Un *PL* qui a la main s'exécute jusqu'à ce qu'il n'ait plus de message d'entrée à traiter. Ensuite, il se bloque. Lorsqu'un *PL* bloqué reçoit un message d'entrée, il est mis dans la liste des *PL*s en attente. Ensuite, celui en tête de liste est activé pour traiter ses messages d'entrées.

Cet algorithme est facilement réalisé à l'aide d'une liste circulaire des descripteurs des *PL*s. Bien que cet algorithme soit facile à réaliser, il ne prend pas en considération les interactions temporelles des événements entre les *PL*s.

2. **Ordonnancement *MVT* ("Minimum Virtual Time)"**

Afin d'assurer l'avancement régulier et sûr de la simulation, il est judicieux de donner la priorité aux *PL*s ayant la plus petite date virtuelle locale (*LVT*). L'algorithme d'ordonnancement de la date virtuelle minimale choisit toujours les *PL*s ayant les plus petites *LVT*s. Le *PL* activé exécute un seul message en entrée, puis le processeur ordonnance l'activation d'un autre. Cet algorithme se réalise efficacement en maintenant les descripteurs des *PL*s dans une liste triée par leur *LVT*.

Les algorithmes d'ordonnancement *MVT* avance le *GVT* en avançant la plus petite *LVT* de *PL*s sur chaque processeur. Cet algorithme offre implicitement la collection régulière des fossiles et allège ainsi la consommation de mémoire au cours de la simulation.

3. **Ordonnancement *MMT* ("Minimum Message Timestamp")**

L'algorithme d'ordonnancement *MMT* donne la priorité aux *PL*s ayant la plus petite estampille de message dans sa liste des messages d'entrée. Cette approche d'ordonnancement tente de réduire le nombre de rollback de l'algorithme *MVT*. Car la plus petite estampille de messages en entrée est toujours inférieure ou égale à la valeur de *LVT*. Ainsi, cet algorithme a beaucoup moins tendance à resimuler ses activités que les autres car les *PL*s prennent moins de risques.

De manière similaire à l'algorithme d'ordonnancement *MVT*, cet algorithme peut être réalisé en mémorisant les descripteurs des *PL*s dans une liste triée selon l'estampille des messages d'entrée de chaque *PL*.

8.4.3 Analyse des algorithmes

Le deuxième critère d'ordonnancement (cf. critères 8.4.1) est difficile à remplir. Il résulte de l'efficacité globale du simulateur optimiste à partir de la coopération des *PL*s voisins. Les trois stratégies d'ordonnancement ne satisfont pas ce critère. En effet, il existe des situations de la simulation optimiste où les trois stratègies d'ordonnancement détériorent la performance du système entier.

Par exemple:

- Pour l'algorithme *MMT*, par une mauvaise partition et l'effet du délai de communication, il pourrait arriver que le *PL* ordonnancé sur un processeur exécute d'abord une suite de messages conduisant uniquement à des rollbacks, alors que d'autres *PL*s

sont prêts à émettre des messages d'événement. Par conséquent, cet algorithme ne conduit pas à l'efficacité du simulateur entier.

- L'algorithme MVT est moins précis que celui de MMT car la condition de sélection est moins forte. Alors, comme l'algorithme MMT ne vérifie pas le critère 2 (cf. critère 8.4.1), MVT ne le remplit pas non plus.

- L'algorithme RR fonctionne de manière aveugle. Il n'utilise aucune technique pour rendre le simulateur efficace; par conséquent, il ne satisfait pas non plus le critère 2 (cf. critère 8.4.1).

En résumé, aucun des algorithmes cités ne remplit les critères d'un bon algorithme d'ordonnancement. Une autre approche à ces algorithmes est l'*algorithme d'auto-ordonnancement* qui ordonnance les PLs à partir de l'heuristique de la "qualité de synchronisation" $Q_{synchro}$.

Soient:
n le nombre de processus à ordonnancer sur ce processeur.
list: une liste de taille n,
ordonné selon les valeurs décroissantes de $Q_{synchro}$
begin
WHILE NOT EndofSimulation DO
IF NewComputation(GVT) THEN
$Q_{synchro} := k \cdot \frac{E_d}{E_t \cdot L_r \cdot N_r \cdot M_n}$
Sort(list, $Q_{synchro}$);
ELSE
GetFromHead(PL, *list*);
Schedule(PL);
InsertInQueue(PL,list);
END IF;
END WHILE
end.

Figure 8.5: *Algorithme d'auto-ordonnancement des PLs*

8.4.4 Ordonnancement selon la "qualité de synchronisation"

Une approche heuristique conforme aux critères ci-dessus, pour choisir le PL à ordonnancer, est délicate. Notre algorithme est basé sur l'heuristique suivante:

Un PL ayant une grande valeur de $Q_{synchro}$ a la priorité pour être ordonnancé car son activité accroît l'efficacité du PL et, par conséquent, l'efficacité du simulateur optimiste.

Ainsi la priorité d'ordonnancement d'un processus logique PL est basée sur ses paramètres de contrôle qui déterminent la valeur de $Q_{synchro}$ qualité de synchronisation. L'algorithme satisfait alors le critère 2, par définition de $Q_{synchro}$.

L'algorithme d'ordonnancement est présenté dans la figure 8.5. L'algorithme est réalisé à l'aide d'une liste des descripteurs des PLs selon la valeur de $Q_{synchro}$ sur un processeur. La PL ayant la plus grande valeur de $Q_{synchro}$ est prioritaire. Pour éviter la famine, une fois qu'un PL finit son exécution entre deux calculs de GVT, il est placé en queue de la liste. Ensuite, le prochain PL dont $Q_{synchro}$ est la plus grande est ordonnancé. Ainsi, tous les processus logiques ont une chance équitable d'être ordonnancés (les critères 1 et 3 sont satisfaits). A chaque nouvelle valeur de GVT, la valeur de $Q_{synchro}$ de chaque processus change, et la liste des PLs doit être triée. En fait, chaque processus logique s'ordonnance compte tenu de ses performances locales, d'où le nom de la stratégie d'auto-ordonnancement.

En résumé, l'algorithme d'ordonnancement satisfait les trois critères ci-dessus. Par conséquent, il est un "bon algorithme" d'ordonnancement des PLs sur un processeur dans un simulateur optimiste. La figure 8.5 présente la structure algorithmique d'auto-ordonnancement des PLs de la simulation répartie.

8.5 Conclusion

La simulation dynamique répartie permet, d'une part, de réduire le coût global des rollbacks et, d'autre part, de limiter la progression des dates virtuelles locales des PLs afin de réduire la différence temporelle entre le processus logique le plus rapide et le plus lent dans le simulateur. Elle offre également un moyen de contrôle adaptatif des paramètres de contrôle du simulateur au cours de l'exécution. Ainsi, on arrive à réduire au minimum le nombre de rollback dans le simulateur et à le rendre stable.

Les principaux avantages qu'apporte le mécanisme dynamique de la distorsion temporelle peuvent être résumés comme suit:

1. réduction de place mémoire utilisée en ayant recours à une période adaptative de sauvegarde d'états. La période varie dynamiquement au cours de la simulation selon la qualité de synchronisation de chaque PL

2. équilibre de la simulation en utilisant une taille adaptative de fenêtre temporelle pour limiter les risques. Une simulation répartie est équilibrée lorsque les dates virtuelles locales LVTs des PLs du simulateur avancent à des vitesses proches. Cette approche vise à:

   - empêcher un PL d'être trop en avance sur ses voisins; par conséquent, elle limite le nombre et surtout la longueur des rollbacks;
   - éviter la situation de famine qui pourrait résulter d'un grand flux de messages simulés, mais non consommés.

3. ordonnancement équitable des PLs dans un environnement de ressources (processeurs) limitées sur une plate-forme où le nombre de ressources (processeurs) disponibles est inférieur au nombre de PLs dans la simulation. Chaque PL est ordonnancé selon sa qualité instantanée de synchronisation; d'où la stratégie d'auto-ordonnancement des PLs.

Hormis ces avantages majeurs, on peut encore retenir:

- l'introduction de la notion de simulateur stable,
- l'établissement des critères d'un ordonnanceur spécifique à la simulation optimiste,
- l'approche dynamique dans la simulation répartie par événements discrets.

Partie III

Etude du contrôle global de la simulation optimiste

Chapitre 9

Les algorithmes de détermination de GVT

Résumé

La base de notre approche est la simulation optimiste où la méthode de synchronisation est la *distorsion temporelle* ("time warp") [Jef85]. Le simulateur est composé d'un ensemble de processus logiques *PLs* répartis sur des processeurs. Chaque *PL* traite les événements de son sous-modèle et avance son horloge locale virtuelle *LVT* indépendamment des autres. La coopération entre les différents processeurs pour résoudre un problème complexe peut produire des erreurs de causalité. Dès qu'une erreur de causalité survient, le *PL* qui la détecte restaure un état cohérent préalablement sauvé, annule les actions entreprises entre le dernier état cohérent et le point de perte de synchronisation, et reprend le cours normal de son exécution. En théorie, chaque *PL* peut sauver autant d'information qu'il veut mais en pratique, la mémoire utilisée pour sauver ses états est limitée et a besoin d'être récupérée pour éviter l'effet de débordement. La *date virtuelle globale ou GVT* ("global virtual time") détermine les états de l'ensemble du système qui peuvent être effacés sans mettre en cause le bon fonctionnement du simulateur. Le but de ce chapitre est de définir les différents sens de *GVT*, d'exprimer la difficulté à déterminer cette valeur et surtout de présenter un état de l'art des algorithmes de GVT dans un simulateur optimiste.

9.1 Introduction

Le contrôle local de la distorsion temporelle laisse certaines questions sans réponse. Comment peut-on s'assurer malgré tous les rollback que le simulateur avance globalement? Comment peut-on détecter la terminaison globale? Comment peut-on valider les opérations d'Entrée et Sortie avec les rollback? Et comment peut-on éviter le débordement de mémoire quand le contrôle local désire sauvegarder des informations?

Le contrôle global résout ces questions. Le concept central du contrôle global se base sur la date virtuelle globale *GVT* qui est une grandeur instantanée et globale du

simulateur. A un instant donné, la GVT représente le minorants des dates de rollback possibles. Les actions effectuées à une heure simulée antérieure à GVT ne seront jamais défaites. Les états et les messages sauvés de date inférieure à GVT sont considérés comme définitifs. Ils peuvent être oubliés. De même, les E/S antérieures à GVT peuvent être réalisées effectivement.

Pour faire progresser GVT, on exécute à intervalles de temps réguliers un algorithme réparti qui estime GVT à partir des dates virtuelles locales LVT de chaque PL et des dates d'émission des messages en attente de traitement. Ce chapitre présente les algorithmes qui déterminent une estimation de GVT.

On n'est pas obligé de conserver l'état des processus après chaque évènement. On peut se contenter de dates de reprise plus ou moins espacées. Evidemment, plus les dates de reprises sont espacées, plus un rollback est coûteux en temps de calcul; car il faut *défaire* et *re-simuler* les actions sur une longue période.

Ce chapitre expose les différentes notions relatives à GVT. La section 2 introduit la notion d'état global par un exemple. La section 3 présente la définition de GVT, les critères d'un algorithme de GVT, son importance et les difficultés pour l'estimer. La section 4 aborde les problèmes des messages en transit. Les sections 5, 6, 7 abordent le problème de report instantané d'information pour déterminer un état global cohérent à partir duquel GVT est approché; la section 5 présente les algorithmes à contrôle centralisé. La section 6, les algorithmes à structure hiérarchique et la section 7 l'algorithme à contrôle réparti. La section 8 conclut par une synthèse qui présente les forces et les faiblesses des algorithmes de GVT.

9.2 Etat global cohérent

Pour cerner le problème de détection d'un état global, on peut utiliser la métaphore de l'observation d'une scène dynamique comme un vol d'oiseaux par un groupe de photographes [CL85]. La scène est tellement vaste qu'elle ne peut être prise par un seul photographe. Plusieurs photographes se chargent alors de prendre chacun une partie de la scène. On admet que les photographes ne perturbent pas le phénomène observé. L'objectif est de déterminer à tout instant le stade d'avancement du vol ou son état global cohérent.

9.2.1 Caractéristique d'un état global cohérent

La détermination de GVT est subordonnée au problème de détection d'un état global cohérent. Un état global cohérent s'obtient en prenant conceptuellement une vue instantanée de l'ensemble du système. Il représente une vision *correcte* instantanée de l'ensemble du système. Un état global cohérent est formé de la réunion de l'état local de chacun des PLs qui forment le système logique et de l'état de chacun des canaux qui relient ceux-ci. L'état d'un PL est défini par ses variables. L'état d'un canal est défini par la séquence de messages dont il assure momentanément le transit.

Il n'est pas possible d'effectuer une observation "simultanée" de l'état des différents processus et canaux dans un système réparti. La réalisation d'une observation cohérente

qui reflète un état global constitué des états locaux des différents processus logiques du système, pris à des instants physiques différents mais de manière à rendre une information utile sur l'état du système dans son intégralité, constitue donc un problème classique, connu, sous le nom de détection d'un état global cohérent.

L'obtention d'un état global cohérent est très utile car elle dérive des *propriétés stables* du système. Il s'agit des propriétés qui une fois vérifiées, restent vraies. Le processus de contrôle chargé de détecter une telle propriété calcule un état global, vérifie la propriété et recommence si elle n'est pas vérifiée.

Formellement, un prédicat P est une propriété stable du système si *pour tout état S' atteignable à partir d'un état S:* $P(S) => P(S')$.

Un état global cohérent permet de détecter les propriétés qui caractérisent l'exécution (passage dans un état particulier, vérification d'assertions, terminaison, impasse), d'effectuer des mesures de performances et de capter des états pouvant servir de point de reprise en cas de défaillances ultérieures de l'exécution.

9.3 Que désigne-t-on par GVT ?

Le contrôle global dans la simulation optimsite se base sur la valeur de GVT. Cette section présente la valeur abstraite et la valeur concrète (opérationnelle) de GVT ainsi que sa définition et la difficulté de son estimation.

9.3.1 GVT valeur abstraite et opérationnelle

Le terme GVT, *date globale virtuelle*, désigne deux valeurs différentes dans une simulation optimiste. La *première*, $GVT(t)$, est une valeur instantanée et abstraite. Conceptuellement, elle se calcule à un instant t en bloquant d'abord tous les PLs pour prendre le minimum parmi (1) la plus petite valeur des $LVT(t)$, (2) la plus petite estampille des messages en transit dans un canal et la plus petite estampille des messages non-traités dans le processus. Les messages en transit sont les messages en attente dans les canaux. La valeur abstraite $GVT(t)$ représente le stade d'avancement de la simulation. Tout message d'événement simulé avant $GVT(t)$ est irrévoccable, cohérent et correct. Par contre, tout événement simulé après cette valeur est susceptible d'être annulé. Il est impossible de déterminer la valeur exacte de $GVT(t)$ au cours d'une exécution répartie asynchrone à cause du report instantané d'informations. Dés lors on se contente de la *seconde valeur* GVT qui se rapporte à une approximation de $GVT(t)$; elle représente une borne inférieure de la valeur exacte $GVT(t)$.

Remarque 10 (Notation de $GVT(t)$ et GVT)
Par convention, nous notons la valeur conceptuelle de GVT, instantanée et exacte par $GVT(t)$ où t désigne l'instant réel précis auquel on détermine la valeur exacte de GVT. La valeur opérationnelle estimée est noté GVT.

Définition 6 (Définition de $GVT(t)$)
La date globale virtuelle exacte $GVT(t)$ [Jef85] à un instant réel t, est le minimum entre:

- *des dates virtuelles locales $LVT(t)$ de chaque PL à l'instant réel t;*
- *des dates virtuelles d'expédition des messages qui ne sont pas encore traités à l'instant réel t.*

Introduisons quelques notations pour exprimer $GVT(t)$ sous forme d'équation; soient:

P: l'ensemble des PLs,

$ts(m)$: l'estampille du message m,

$LVT_i(t)$: la date virtuelle locale du PL_i à l'instant réel t,

$T_{i,j}(t)$: l'ensemble des messages en transit à l'instant t dans le canal uni-directionnel qui va du PL_i au PL_j;

$T_j(t)$: l'ensemble des messages en attente vers le PL_j à l'instant réel t. Il s'exprime par:

$$T_j(t) = \bigcup_{\forall i: PL_i \in P} T_{i,j}(t)$$

$T(t)$: l'ensemble des messages en attente de traitement à l'instant réel t dans le système entier. Il est défini par:

$$T(t) = \bigcup_{\forall j: PL_j \in P} T_j(t)$$

$LM_j(t)$:est le minimum local du PL_i à l'instant t. Il est défini comme suit:

$$LM_j(t) = \min\{LVT_j(t), \min\{ts(m), m \in T_j(t)\}\}$$

La valeur de $GVT(t)$ à l'instant t est le minimum des minima locaux à l'instant t. $GVT(t)$ peut donc s'exprimer sous la forme suivante:

$$GVT(t) = \min\left\{\min_{PL_j \in P}\{LVT_j(t)\}, \min\{ts(m), m \in T(t)\}\right\}$$

$$GVT(t) = \min_{PL_j \in P}\{LM_j(t)\}$$

$GVT(t)$ [Jef85] est une valeur instantanée *monotone non décroissante.* Elle désigne la borne inférieure des dates virtuelles qu'un rollback peut atteindre.

Définition 7 (*GVT opérationnel*)
La date GVT se définit comme la plus petite valeur:

- *du minimum des dates virtuelles locales LVT;*
- *du minimum de toutes les dates des messages en transit;*
- *du minimum des estampilles des messages non traités dans la file d'attente d'entrée de chaque processus logique.*

La différence entre la détermination de GVT et de $GVT(t)$ provient du fait que la valeur exacte de la première est déterminée à un instant réel donné, tandis que la seconde, approchée, se calcule sur un intervalle de temps.

La valeur de GVT doit être régulièrement revue car la performance du systéme dépend de la fréquence choisie pour approcher $GVT(t)$. Quels sont les critères de conception d'un algorithme de GVT?

9.3.2 Critères ou qualité de GVT

Le critère principal de la conception d'un algorithme de GVT [Lin94] est sa simplicité. Un simulateur englobe plusieurs algorithmes répartis [Jef90]: les algorithmes de gestion de mémoire, les algorithmes de terminaison, les algorithmes de tolérance aux erreurs, les algorithmes de sauvegarde des états et les algorithmes d'annulation ("rollback"). La conception de ces algorithmes dépend de la qualité de l'algorithme de GVT.

Une bonne approximation GVT à l'instant t doit être assez proche de sa valeur abstraite exacte $GVT(t)$. Un bon algorithme de GVT satisfait deux propriétés: la *sûreté* et la *vivacité*. La sûreté concerne l'exactitude de l'estimation de GVT alors que la vivacité assure que la différence temporelle entre la valeur estimée GVT et la valeur instantanée $GVT(t)$ est bornée.

Condition 3 (Propriété de sûreté)
Soit GVT la valeur approchée de GVT(t) par un algorithme de GVT à un instant t. Cet algorithme est sûr si et seulement si: $GVT <= GVT(t)$.

Condition 4 (Propriété de vivacité)
L'algorithme est vivant: Si $t_{début}$ *désigne la date où l'on commence à estimer* $GVT(t)$ *et* t_{fin} *la date de la fin de cette estimation, la valeur calculée GVT doit satisfaire:* $GVT(t_{début}) <= GVT <= GVT(t_{fin})$.

La condition de sûreté de l'algorithme garantit que la valeur estimée GVT n'est jamais plus grande que sa valeur exacte $GVT(t)$, alors que la condition de vivacité garantit que si la valeur exacte augmente, sa valeur estimée croît aussi. La condition de sûreté exprime que les valeurs estimées de GVT sont exactes, tandis que celle de vivacité (réutilisant la propriétés de vivacité) exprime que ces valeurs estimées sont monotones non décroissantes. En somme, un *bon algorithme de GVT* doit fournir des *estimations monotones non décroissantes de GVT*.

9.3.3 Importance de *GVT*

La valeur estimée de *GVT* permet:

- de valider la simulation jusqu'à la date *GVT*;
- d'effectuer la collection des fossiles (la récupération de la mémoire des événements corrects);
- de détecter la terminaison de la simulation et faire le rapport de la simulation;
- de prouver que l'algorithme de synchronisation termine, car *GVT* est une fonction monotone non décroissante;
- de gérer les erreurs.

Récupération de l'espace mémoire

La valeur estimée *GVT* intervient dans la gestion des mémoires des listes suivantes:

- la liste des états récemment sauvés ("stateQueue");
- la liste des anti-messages à la sortie ("outputQueue");
- la liste des messages déjà traités en entrée ("processedIQ");
- la liste des messages non-traités en entrée ("unProcessedIQ").

Les deux dernières listes sont disjointes; leur union donne la liste des messages en entrée du *PL* "inputQueue".

Les trois premières structures utilisées pour concevoir le rollback sont gérées de manière similaire. Un message dans une file d'attente dont l'estampille est inférieure à *GVT* est détruit car il est impossible de revenir en arrière au-dessous de *GVT*.

La gestion de la dernière structure de liste est le problème commun de contrôle de flux propres aux systèmes répartis. Si la liste des messages en entrée est pleine à la réception d'un nouveau message, la distorsion temporelle libère la place mémoire par un renvoi du message à l'expéditeur ("sendback"), une annulation en arrière ("cancelback") ou un rollback artificiel ("artificial rollback").

Détection de la terminaison normale

Rappelons que lorsqu'un processus logique a exécuté tous ses messages, il est terminé et sa date virtuelle locale prend la valeur $+\infty$. *GVT* détecte la terminaison normale de la simulation quand elle prend la valeur $+\infty$; dès lors, aucun message n'est plus transmis dans le simulateur.

Gestion des erreurs

Quand un processus logique commet une erreur, son état marque la présence d'une erreur. Mais il peut continuer son exécution malgré la présence de l'erreur dans l'espoir qu'un rollback corrige l'erreur. Si l'erreur persiste et apparaît dans un des états fossiles (état dont l'estampille est inférieure à *GVT*), aucun mécanisme ne peut corriger cette erreur et le processus est arrêté.

Opération d'Entrées et Sorties

Les requêtes d'entrée sont mémorisées et leur traitement différé. Les activités physiques de sortie sont uniquement exécutées pour les messages de requête dont l'estampille est inférieure à GVT. GVT permet de lancer l'étude partielle de paramètres de sortie de la simulation.

9.3.4 Difficulté d'estimation de GVT

Avant de présenter les différents algorithmes de GVT, il est nécessaire de se convaincre de la difficulté du calcul de GVT. En fait, le calcul de GVT nécessite la connaissance de l'état global du système à chaque instant où l'on désire connaître sa valeur. Un PL désireux d'obtenir un *état global cohérent* doit intercepter instantanément l'état de chacun des autres processus et l'état de chacun des canaux du système. L'algorithme d'obtention de la perception instantanée des observations (état global cohérent) dans un système réparti n'est pas trivial, car l'objet que l'on observe peut se modifier durant l'observation, mais il ne faut en aucun cas que l'observation altère l'état global cohérent. C'est le problème de *report instantané d'informations.* Concrètement, pour déterminer GVT il faut:

- visiter tous les PLs et observer leur date virtuelle locale LVT;
- visiter tous les canaux et observer l'estampille $t(m)$ des messages m en transit;
- rassembler toutes ces observations de façon à ce que l'observation globale obtenue soit cohérente.

Le calcul de GVT est complexe dans un système réparti car il est difficile d'obtenir un état global cohérent. En particulier, il pose deux problèmes: la prise en considération des messages en transit et le problème de report instantané des informations pour déterminer GVT.

La section suivante présente la résolution des messages en transit et les autres sections seront consacrées au problème du report instantané des informations d'estimation de GVT.

9.4 Problèmes des messages en transit

Certains algorithmes de GVT font abstraction des messages en transit pour calculer GVT à partir des LVTs et des messages non-traités dans les listes des messages en entrée de chaque PL; mais en considérant les messages en transit, la réception d'un message en transit peut mettre en cause la valeur de GVT estimée. Pour identifier les messages en transit, chaque PL expéditeur garde en mémoire les messages émis jusqu'à ce qu'il obtienne de son récepteur une quittance de réception du message. Ensuite l'expéditeur récupère la place mémoire des messages émis et reçus.

Les algorithmes efficaces du traitement des messages en transit tiennent compte des propriétés d'interconnexion du réseau. Nous décrivons ci-après les propriétés des canaux et les approches pour maîtriser le problème des messages en transit.

9.4.1 Propriétés PAPS de canaux

Dans plusieurs cas, les canaux du réseau satisfont la condition PAPS: premier arrivé, premier servi.

Condition 5 (condition PAPS)
L'interconnexion d'un réseau satisfait la condition PAPS si: lorsque PL_i envoie à PL_j le message m et plus tard le message m' de PL_i à PL_j, alors PL_j reçoit m avant m'.

Preiss [Pre89] a observé que cette condition est nécessaire pour que l'algorithme de circulation du jeton sur un anneau ("token ring") fonctionne correctement.

En fait, pour garantir l'exactitude des algorithmes de GVT, les canaux de communication doivent vérifier la propriété plus forte suivante:

Condition 6 (condition d'interconnexion transitive PAPS)
L'interconnexion d'un réseau satisfait la condition transitive de PAPS [BCC91] si PL_i envoie le message m à PL_k et plus tard un autre message m' est envoyé à PL_j et que PL_j envoie un message $m"$ à PL_k après avoir reçu m', alors PL_k reçoit m avant $m"$.

La condition d'interconnexion transitive PAPS porte sur l'ensemble des canaux, contrairement à la condition PAPS qui se rapporte à un canal. Lorsque la condition d'interconnexion transitive est satisfaite, on dit que l'ensemble des messages du système satisfait la *propriété d'ordre causal* [RST91]. L'ordre causal définit un ordre sur les réceptions de messages plus fort que celui imposé par les canaux PAPS [Tel91]. Donc, si la propriété d'ordre causal est satisfaite, chaque canal du réseau satisfait la condition PAPS. Un réseau dans lequel cette propriété est garantie résout le problème des messages en transit. Ainsi, GVT se calcule à partir des LVTs et des messages non-traités dans la liste des messages d'entrée. Le problème se résume à "la collection instantanée" de ces informations.

La condition transitive de PAPS est vérifiée quand les primitives de messages sont réalisées sur une architecture à mémoire partagée. Elle est aussi satisfaite si l'interconnexion du réseau ne fournit pas plusieurs routages, comme dans une structure d'arbre ou dans un anneau.

9.4.2 Messages avec accusé de réception

Une manière de résoudre le problème des messages en transit lorsque la condition transitive de PAPS n'est pas satisfaite est d'utiliser les *messages avec accusé de réception.* Le principe du message avec accusé de réception est le suivant: lorsqu'un message m est envoyé du processus PL_i au processus PL_j, PL_j doit, dés la réception du message m, envoyer un message d'accusé de réception à PL_i. Ainsi, PL_i peut facilement déceler si m est en transit, c'est-à-dire qu'il n'a pas encore reçu la quittance de réception de m de la part de PL_j.

Un désavantage de l'utilisation des messages avec accusé de réception est qu'elle double le trafic de messages dans le réseau ainsi que l'occupation en mémoire des messages expédiés.

En pratique, pour réduire le nombre de messages vers l'expéditeur, il faut veiller à ce que le nombre de messages envoyés soit minimal et qu'un expéditeur n'attende pas indéfiniment la quittance d'un message. Un numéro de séquence est associé à chaque message. Ces numéros sont croissants dans un canal de communication. Le récepteur garde en mémoire les numéros manquants et envoie à l'expéditeur le dernier numéro reçu dans l'ordre consécutif pour l'informer de la réception de tous ses messages précédents (y compris celui du dernier numéro de séquence). Ce principe est utilisé par les algorithmes à modèle centralisé pour maîtriser le problème des messages en transit lorsque la condition de transitive de PAPS n'est pas satisfaite [LL90b].

9.4.3 L'approche de Samadi

Samadi [Sam85] propose une approche sans accusé de réception pour maîtriser le problème des messages en transit. L'idée est que chaque message comporte un numéro. Lorsqu'un processus logique PL_i reçoit un message, il envoie à chacun de ses voisins PL_j le plus petit numéro de message manquant. Ainsi, il envoie implicitement un message d'accusé de réception pour tous les messages dont le numéro est inférieur au nombre envoyé. Le processus logique PL_j suppose que ce message et tous ceux dont le numéro est plus grand sont des messages encore en transit. Pour les messages en transit, le processus PL_j est capable de déterminer la borne inférieure de leur estampille car il conserve la liste des messages sans accusé réception. Le PL a le minimum local des estampilles de message et leurs numéros à traiter.

9.5 Algorithmes à contrôle centralisé

Dans cette classe d'algorithmes, un gestionnaire est chargé de tenir à jour les états locaux cohérents des canaux et des PLs du système réparti. Tous les PLs (sites, processeurs) lui communiquent leurs états locaux cohérents.

9.5.1 Principe de fonctionnement

Soit n PLs dans le système logique numérotés de 1 à n et $[Start_i, Stop_i]$ un intervalle de temps réel de l'estimation de GVT par le PL_i. Nous exigeons que l'intersection de ces intervalles ait au moins une date commune appelée RTM ("real-time moment") et que chaque PL_i puisse déterminer son minimum local LM_i.

L'algorithme classique [Sam85] d'estimation de GVT s'opère sur un intervalle de temps réel $[Start_i, Stop_i]$ en cinq phases:

1. *diffusion du signal "début de calcul de GVT"* Le gestionnaire central envoie un message "début de calcul de GVT" à tous les autres PLs. En fait, la date de réception de ce message par chaque PL_i correspond à sa date du début réel de l'évaluation $Start_i$.

2. *collection des réponses par le gestionnaire* Chaque PL_i répond au message du gestionnaire en lui envoyant un message d'accusé de réception. Lorsque le gestionnaire a reçu la réponse de tous les PLs, il initialise RTM à l'instant réel t.

3. *diffusion du signal "arrêt pour calculer GVT"* Le gestionnaire envoie un message d'arrêt à chaque PL_i pour geler leur état. A la réception du message d'arrêt, PL_i met à jour son instant $Stop_i$. C'est dès cet instant réel que le calcul de GVT commence.

4. *collection des minima locaux par le gestionnaire* Chaque PL_i répond au message d'arrêt en envoyant un message comportant sa date locale minimum ou LM_i. Après que le gestionnaire a reçu la réponse de chaque PL_i, il détermine la valeur GVT, minimum de tous les LM_is.

5. *diffusion de la nouvelle valeur de GVT*

   Le gestionnaire diffuse le message de mise à jour de GVT à tous les autres PLs. La diffusion de GVT signale la fin d'un cycle d'estimation de GVT et libère tous les PLs gelés en phase 3.

9.5.2 Variantes de l'algorithme à contrôle centralisé

Passons ci-après, en revue, les divers algorithmes à contrôle centralisé proposés pour déterminer GVT.

Jefferson [Jef85] a proposé un algorithme en cinq phases où un PL donné du modèle simulé joue le rôle du gestionnaire central.

Lin et Lazowska [LL90b] ont proposé un algorithme similaire à celui de Samadi. L'algorithme fonctionne dans un système asynchrone sans accusé de réception de messages explicite. Mais il comporte deux inconvénients: d'une part, l'approximation de la valeur de GVT n'est pas optimale [Mat93] et, d'autre part, il nécessite une structure de données qui permette de mémoriser la suite des numéros manquants et la valeur des minima locaux des estampilles envoyées.

Lin [Lin94] propose une version améliorée de l'algorithme de Lin et Lazowska [LL90b] qui fonctionne en cinq phases. Il propose un algorithme simple et efficace avec la propriété PAPS de communication des canaux pour résoudre le problème des messages en transit. L'apport de cet article réside dans la preuve formelle d'exactitude du modèle centralisé et l'étude de sa complexité.

Bauer et Sporrer [BS92] ont proposé une version améliorée de l'algorithme de base où chaque PL envoie périodiquement des messages d'événement au gestionnaire central qui initialise le calcul de GVT à partir des informations reçues. Ensuite, le gestionnaire diffuse à des intervalles de temps fixes sa nouvelle valeur GVT à tous les PLs.

D'souza [DFW94] propose une variante de l'algorithme de Bauer et Sporrer [BS92]: un algorithme passif *pGVT* où la communication PAPS des messages n'est pas garantie. Les messages sont envoyés avec un accusé de réception. L'algorithme est dit passif parce que le gestionnaire central calcule la valeur de GVT à partir des informations rapportées par les PLs eux-mêmes plutôt qu'à la demande explicite du gestionnaire central. L'amélioration visée par l'auteur est de minimiser les reports d'information de calcul de GVT

de chaque PL. Une méthode statistique est proposée pour que chaque processus sache quand reporter au gestionnaire sa nouvelle information en vue du calcul de GVT.

9.5.3 Analyse des algorithmes à contrôle centralisé

La présence d'un gestionnaire implique la gestion d'un réseau supplémentaire de contrôle utilisé exclusivement pour déterminer GVT.

Le calcul de GVT nécessite une *forte synchronisation* où tous les PLs doivent être gelés pour déterminer le minimum local selon la phase 3 du fonctionnement.

Une difficulté majeure des algorithmes à contrôle centralisé provient du fait que le calcul d'un cycle de calcul GVT se fait sous l'initiative du gestionnaire central qui interroge tous les PLs. L'avancement de GVT s'ensuit. Une grande fréquence d'estimation de GVT (temps réel entre deux calculs de GVT) régulière n'entraîne pas un avancement régulier de GVT. Alors qu'une petite fréquence pourrait introduire un grand décalage entre la valeur estimée GVT et la valeur exacte de $GVT(t)$ [Jef85], ce qui augmente l'utilisation de mémoire avec le risque de rupture de la simulation.

Les phases (2) et (4) comportent des collections d'information où le gestionnaire reçoit séquentiellement un message de chaque PL. Cet algorithme est d'ordre $O(n)$ où n désigne le nombre des PLs, alors que les messages de diffusion aux phases (1), (3) et (5) dépendent de l'architecture cible et de la topologie du réseau des PLs. Sur une architecture paralléle qui supporte la diffusion de messages, ces phases sont d'ordre $O(1)$. Mais si ces messages sont envoyés en série, alors la méthode s'exécute en $O(n)$. Selon l'architecture, cet algorithme classique envoie de $3n$ à $5n$ messages.

9.6 Algorithmes à structure hiérarchique

Ces algorithmes sont conçus pour améliorer les algorithmes à contrôle centralisé. Ils supposent une configuration particulière du réseau des PLs, par exemple une structuration en arbre pour former l'arbre des *états cohérents locaux* ou en anneau circulaire à plusieurs niveaux.

9.6.1 Algorithmes à structure de graphe

L'idée est de structurer l'ensemble des PLs du simulateur en arbre ou graphe particulier. Tout noeud de l'arbre (autres que les feuilles) gère le graphe de ses noeuds-fils. Ainsi l'ancêtre commun des PLs ou la racine de l'arbre (ou graphe) détient l'état cohérent global. La racine de l'arbre (graphe) détermine la valeur de GVT du systéme qui remonte depuis les feuilles de l'arbre.

Principe

Cette classe d'algorithmes est conçue pour abaisser le nombre de messages transmis dans le calcul de GVT de l'ordre de $O(n)$ à $O(log\ (n))$. Pour cerner le principe de fonctionnement,

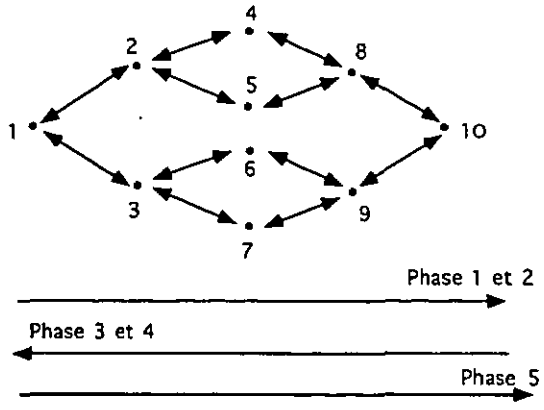


Figure 9.1: graphe équilibré et symétrique ("binary-tree-like forwarding")

nous supposons que les PLs sont structurés sous la forme d'un *graphe binaire équilibré et symétrique*("binary-tree-like forwarding") selon la figure 9.1.

Il est possible de combiner les phases de collection et de diffusion à l'aide d'une structure de graphe binaire équilibré et symétrique pour faire suivre les messages. Le PL_i diffuse ses informations aux PL_{2i} et PL_{2i+1} où $2i+1 < N$. La phase de la collection peut se combiner avec la phase d'évaluation partielle de GVT en remontant le graphe bianire (parcours dans le sens inverse); ainsi le PL_i estime LVT_i des processus logiques PL_{2i+1} et PL_{2i+2} et l'envoie au processus logique $PL_{i/2}$ si $(i > 0)$. En utilisant le graphe binaire équilibré et symétrique pour capturer et diffuser les informations, le temps d'exécution de l'algorithme de GVT descend à l'ordre $O(log\ (n))$.

En effet, cette nouvelle approche combine les phases 1 et 2 du modèle à contrôle centralisé en 1, puis les phases 3 et 4 en 2 en remontant le graphe binaire vers la racine lors du parcours en sens inverse. Elle est une amélioration des algorithmes à contrôle centralisé où les phases de collection et de diffusion d'information de l'algorithme à contrôle centralisé sont combinées en une seule.

Variantes de l'algorithme à structure de graphe

Nous étudions ci-aprés les différentes variantes de l'algorithme en structure de graphe.

Bellenot [Bel90] fut probablement le premier à utiliser une structure d'arbre binaire pour initialiser, calculer et diffuser la valeur de GVT. Il aborde le problème en mémorisant des connaissances dans les messages de contrôle qui parcourent la structure du graphe binaire symétrique équilibré (cf. figure 9.1). L'algorithme [Bel90] comporte une étape de pré-traitement qui sert à placer l'ensemble des PLs sur les noeuds du graphe. La valeur de GVT est calculée en parcourant le graphe binaire dans les deux sens.

Baldwin et al. [BCC91] ont par la suite généralisé l'approche pour une structure d'arbre binaire à deux niveaux. L'apport de leurs travaux réside dans la preuve de la sûreté et de la vivacité pour valider l'exactitude GVT lorsque l'ensemble des PLs sont structurés sous la forme d'arbre binaire.

Briner [Bri90] et *Reynolds* [Rey91] ont proposé une adaptation matérielle ("hard-

ware") de l'algorithme à structure d'arbre sur une machine multiprocesseur à mémoire partagée pour déterminer GVT.

9.6.2 Algorithmes en anneau

Le but de cette approche est d'estimer GVT par un algorithme réparti sans gestionnaire central et sans bloquer les PLs.

Principes

L'idée est de définir un message de contrôle spécial ou *jeton* qui circule dans l'ensemble des processus et des canaux supposés connectés selon une topologie en anneau. Cette configuration nécessite la création et la gestion d'un réseau supplémentaire à celui du modèle sous-jacent. L'anneau peut être défini statiquement ou configuré dynamiquement.

Le rôle du jeton est:

- de récolter la plus petite date des PLs et canaux visités pour déterminer la prochaine valeur de GVT; et
- de propager la date courante de GVT aux autres PLs du système.

Chaque PL contient deux variables locales:

- LVT: la date virtuelle locale, elle varie avec les événements reçus et peut être désynchronisée par un rollback;
- $LVTmin$: la date enregistrée au dernier passage du jeton.

Le jeton est un enregistrement $< GVT, tcandidat, idcandidat, couleur >$ où GVT désigne la date virtuelle globale. $tcandidat$ désigne le minimum instantané des dates virtuelles recueillies par le jeton; c'est le candidat potentiel futur à GVT. $idCandidat$ désigne le PL candidat qui détient ce minimum; $couleur$ désigne l'identité du jeton quand plusieurs jetons sont simultanément en action. Le jeton ($tcandidat$) contient la plus petite date des PLs et des canaux visités.

A la réception du jeton $< GVT, tcandidat, idcandidat, couleur >$ par un PL:

- soit le PL est le propriétaire du jeton (PL=$idCandidat$); il vient alors de faire un tour complet et possède par conséquent la plus petite date de tous les canaux et PLs visités; il affecte à GVT la valeur de $tcandidat$.
- soit le PL n'est pas propriétaire du jeton; il propage alors la valeur de GVT et met à jour les caractéristiques du jeton ($tcandidat, idcandidat$).

La structure algorithmique de chaque PL à la réception d'un jeton

$< GVT, tcandidat, idcandidat, couleur >$

est donnée dans la figure 9.2.

```
begin
    receive (<GVT, tcandidat, idcandidat, couleur>)
    IF (idcandidat = j) THEN
        le jeton vient de faire un tour complet
        GVT := tcandidat;
        tcandidat := LVTmin;
    ELSE
        IF LVTmin <tcandidat THEN
        meilleur candidat de GVT
            tcandidat := LVTmin;
            idcandidat := j;
        END IF
    END IF
    LVTmin := LVT;
    UpdateFossileCollection(GVT)
    sendToNext (<GVT, tcandidat, idcandidat, couleur>)
end.
```

Figure 9.2: *Structure du jeton circulant à un niveau*

Variantes de l'algorithme en anneau

Nous étudions ci-après les différentes variantes de l'algorithme en structure d'anneau. Nous mettons l'accent sur les améliorations apportées aux principes et leur réalisation.

Preiss [Pre89] a utilisé le schéma de la circulation du jeton en anneau où les PLs de la simulation sont organisés en anneau circulaire. Le jeton circule continuellement d'un PL à un autre pour déterminer et communiquer la valeur estimée GVT. Cette approche est similaire à l'algorithme de détection d'impasse basé sur la circulation de jeton en anneau [Mis86].

Baldwin et ses collègues [BCC91] ont par la suite généralisé cette approche en proposant une version de circulation à deux niveaux. La description détaillée de cet algorithme y est fournie à la section 4. Le lecteur intéressé y trouvera la preuve de sûreté et de vivacité de l'approche du jeton circulant sur un anneau à un et à deux niveaux.

Concepcion et Kelly [CK91] vont plus loin en proposant un algorithme généralisé sur un anneau circulaire à plusieurs niveaux. Une observation attentive de cet algorithme révèle [BCC91] que la valeur estimée de GVT n'est pas une fonction croissante. De plus, l'algorithme ne tient pas compte des messages en transit.

Remarque 11
Le jeton est un message de contrôle qui se substitue au gestionnaire central des modèles centralisés. Les algorithmes à contrôle centralisé fournissent la valeur de GVT à un instant donné tandis que l'algorithme du jeton circulant détermine la valeur de GVT dans un intervalle de temps $[0..2Dt]$ où Dt désigne le temps nécessaire pour faire un tour complet de l'anneau. Le premier tour sert à déterminer GVT, alors que le second sert à

propager la valeur déterminée au premier.

9.7 Algorithme à contrôle réparti

Un algorithme GVT à contrôle réparti fonctionne sur le réseau sous-jacent au modèle simulé: le réseau des processus logiques. Chaque PL conserve son autonomie et la panne d'un PL n'affecte pas son ensemble de dépendance, ce qui n'est pas fatal à l'algorithme entier. L'ensemble de dépendance d'un PL est l'ensemble des PL_i desquels PL peut recevoir un message.

Une solution triviale est la *méthode de la duplication* ("replicated method") où chaque PL maintient une copie du graphe des états locaux cohérents des PLs. Ainsi chaque PL possède une vision de l'état global cohérent pour estimer GVT; c'est l'approche de Mattern.

Algorithme de Mattern

Mattern [Mat93] présente un algorithme parallèle pour déterminer un état global cohérent dans un environnement réparti. Une application de l'état global cohérent dans un système de simulation répartie est l'approximation de GVT. L'algorithme ne fait pas d'hypothèse sur la structure topologique du réseau ou la propriété PAPS des canaux de communication.

L'idée est d'utiliser deux couleurs, dont l'une indique si un PL a déjà son état local cohérent et l'autre si le message a été envoyé avant ou après cet état. Ainsi, les messages qui pourraient introduire une incohérence sont facilement reconnus et évités. Les messages en transit sont ainsi capturés par le processus logique destinataire. Le problème des messages en transit est résolu par un schéma de détection de terminaison réparti [MMST91]. L'approximation de GVT est obtenue en appliquant la notion de coupure temporelle qui permet de partitionner les événements produits dans le système en deux sous-ensembles relatifs à "avant" et "après" la coupure. Elle exprime une relation de précédence entre les événements du système. Une coupure C est un ensemble d'événements tels que si $g \in C$ et f précède g alors $f \in C$ [Ray87, page 102].

Le concept de coupure traduit donc les relations entre les événements: la précédence à l'intérieur d'un PL et la causalité entre PLs. A une coupure C il est donc possible d'associer un état global cohérent $S(C)$ du système de la façon suivante:

- pour chaque processus PL_i, l'état de PL_i dans $S(C)$ est l'état résultant du dernier événement de PL_i dans C s'il existe, sinon son état initial;
- pour chaque canal c_j, l'état de c_j dans $S(C)$ est la séquence des messages envoyés appartenant à C et reçus par des événements n'appartenant pas à C.

L'idée principale d'estimation de GVT est d'utiliser deux coupures temporelles pour assurer qu'aucun message en transit ne passe à travers ces deux coupures temporelles. Alors le minimum de l'estampille de tous les messages qui passent à travers la seconde

coupure est facilement déterminé en considérant tous les messages envoyés entre les deux coupures temporelles.

Il y a plusieurs manières de réaliser ces coupures. Mattern a réalisé une version simplifiée [Mat93] qui utilise la circulation du jeton en anneau pour esquisser comment l'algorithme parallèle en deux phases peut être utilisé pour réaliser les deux coupures temporelles dans un réseau, et la détection de la terminaison dans une simulation répartie qui est un cas particulier de l'approximation de GVT. Ce schéma offre un élégant moyen pour détecter la terminaison parallèle de la simulation.

9.8 Critique des algorithmes de GVT

Les algorithmes connus jusqu'à nos jours pour estimer GVT déterminent d'abord un état global cohérent; ils présentent tous au moins l'un des trois désavantages suivants:

- la présence d'un gestionnaire centralisé qui supervise le réseau entier;
- une forte synchronisation qui consiste à bloquer (geler) tous les états du système pour déterminer GVT;
- l'hypothèse d'une topologie particulière (arbre ou anneau) exclusivement utilisée pour calculer GVT.

Les trois moyens ci-dessus introduits subtilement pour déterminer l'état global cohérent vont à l'encontre du concept réparti de contrôle dans un environnement réparti.

Notre objectif dans le chapitre suivant est de proposer un algorithme basé sur un modèle de PL qui élimine ces trois inconvénients à la fois. L'idée principale de l'algorithme consiste à estimer GVT de manière asynchrone en imposant à chaque PL de consulter uniquement les informations de ses prédécesseurs. Chaque PL estime localement son LT (l'équivalent de GVT local) à partir des LVTs de ses prédécesseurs et ses canaux d'entrée.

L'avantage de l'algorithme LT réside au fait que les différents PLs peuvent avoir différentes valeurs de la date locale sûre LT. L'estimation de GVT est dans ce cas, la plus petite valeur des LTs. Ainsi, les PLs dont la valeur de LT est ultérieure GVT sont plus précise et utilise moins de place mémoire, et ceux dont la valeur LT est égale à celle de GVT, utilise la même place mémoire.

Chapitre 10

Estimation locale de GVT: la date locale sûre

Résumé

Ce chapitre propose un algorithme pour déterminer en chaque PL une date locale sûre LT jouant pour ce PL un rôle équivalent à celui de la date virtuelle globale GVT. Cela tient au fait que l'algorithme ne nécessite pas une synchronisation globale de tous les PLs. Le calcul de cette valeur, LT, prend en compte, pour un PL donné, uniquement les PLs qui le précèdent: son *ensemble de dépendance*. Il se distingue des algorithmes existants en deux points essentiels: il ne tient pas compte de l'ensemble des PLs du système pour déterminer la date locale sûre d'un PL. Il est entièrement asynchrone et réparti. De plus, il fournit à chaque PL une valeur LT plus exacte que la valeur de GVT estimé dans le sens ($GVT \leq LT$).

Cet algorithme utilise des messages de contrôle ou *jetons* selon la stratégie sûre pour avancer la date locale sûre d'un PL. Par contre les messages d'événements d'un modèle simulé sont traités selon la stratégie optimiste. Chaque processus logique du simulateur traite ainsi conjointement les jetons pour avancer les dates locales sûres et les messages d'événement pour simuler les activités du modèle.

10.1 Introduction

L'état de l'art des algorithmes de GVT montre le rôle prépondérant que la valeur de GVT joue dans une simulation répartie optimiste. Nous proposons à la place de GVT, pour chaque PL, une variable locale LT désignant *la date locale sûre* (conservative local time) du PL. L'algorithme LT cherche, pour chaque PL, à déterminer localement une valeur équivalente à GVT sans devoir observer un état global cohérent [DFW94] [TG93]. Le calcul de cette valeur prend uniquement en compte les PLs de son ensemble de dépendance. L'*algorithme* LT qui calcule la valeur de LT utilise en arrière-plan une approche sûre modifiée pour faire circuler des jetons depuis les PLs sources. Cet algorithme est beaucoup plus simple que tous les algorithmes existants pour déterminer GVT à partir

d'un état global cohérent. Il nécessite un traitement préalable pour déterminer les composantes fortement connexes du graphe avant de lancer le mécanisme des jetons depuis les *PL*s sources.

La question principale à laquelle ce chapitre tente de répondre se résume ainsi: "*comment peut-on s'assurer que la valeur LT déterminée localement valide les opérations qu'auraient validées le calcul de GVT?*".

La section 2 présente brièvement la synthèse des algorithmes de *GVT* pour relever leurs défauts. La section 3 part d'un exemple pour introduire la notion de date locale sûre. La section 4 présente les hypothèses pour établir l'algorithme *LT*, la section 5, une solution conceptuelle de l'algorithme, la section 6, la description du fonctionnement de l'algorithme. Enfin la section 7 présente la conclusion et les extensions souhaitées dans le futur.

10.2 Critique des algorithmes de GVT existants

Dans leur principe, les algorithmes d'approximation de GVT détectent d'abord un état global cohérent [Tel91]. Une approximation de GVT est une application directe d'état global cohérent ou stable du système [CL85] [LY87]. Il est cependant souhaitable de spécialiser l'algorithme pour augmenter son efficacité. A cet effet, Ted [Tel91] propose plusieurs types de solutions générales selon les modèles de communication, en particulier deux solutions pour les modèles asynchrones:

- la première solution est basée sur un envoi de message suivi d'un accusé de réception et le maintien d'une structure pour représenter l'estampille des messages restés sans réponse; en fait, Samadi et ses collègues [SM87] ont utilisé cet algorithme; tous les efforts de recherche dans cette direction se résument à l'amélioration de cet algorithme;

- la seconde solution conserve une liste de messages reçus et une liste de messages envoyés pour chaque processus. Après avoir collectionné toutes les listes locales, les messages en transit sont déterminés en prenant la différence de ces deux ensembles. C'est le principe utilisé par Lai et Yang [LY87] et Mattern [Mat93] dans leurs algorithmes pour déterminer l'état global cohérent. L'inconvénient majeur de cette solution est qu'elle nécessite beaucoup de mémoire pour conserver les listes locales de messages.

Les efforts d'amélioration des algorithmes de GVT peuvent se résumer en trois catégories:

1. *Algorithmes à contrôle centralisé*

   Dans ces algorithmes, un gestionnaire central défini sur un site est chargé de maintenir à jour les états locaux cohérents des canaux et des *PL*s. Tous les *PL*s (sites) lui communiquent leurs états locaux cohérents sur demande [Bel90], [Bri90] [Lin94] ou de manière passive [DFW94]. Cette solution souffre de plusieurs inconvénients: la

panne du site gestionnaire est fatale à l'ensemble du système, le nombre de messages utilisés est grand et l'autonomie des *PLs* n'est pas exploitée. En outre, on définit un réseau supplémentaire au graphe du modèle simulé pour déterminer *GVT*.

2. *Algorithmes à structure hiérarchique*

   L'ensemble des *PLs* est supposé structuré en arbre: l'arbre des états cohérents locaux. Tout nœud d'arbre (autres que les feuilles) gère un graphe d'évaluation d'état cohérent (ou d'attentes) qui contient les informations nécessaires relatives aux graphes des attentes de ses nœuds-fils. L'ancêtre commun des *PLs* ou la racine de l'arbre comporte l'état cohérent global du système qui sert à calculer le *GVT*. Ce schéma hiérarchique est utilisé par Bauer [BS92] et Bellenot [Bel90].

3. *Algorithmes à contrôle réparti*

   Répartir le contrôle pour déterminer *GVT* consiste à éliminer toute hiérarchie entre les *PLs* (sites). Chaque *PL* conserve son autonomie et la panne d'un *PL* ne touche pas son ensemble de dépendance. Ce qui n'est pas fatal à l'ensemble des *PLs* du système. Rappelons que l'*ensemble de dépendance ED* d'un *PL* est l'ensemble des PL_j desquels *PL* peut recevoir un message de façon directe ou par l'intermédiaire d'autres *PLs*.

   Une solution triviale est de maintenir sur chaque site une copie du graphe des états locaux cohérents des autres *PLs*. Ainsi chaque *PL* a une vision de l'état global cohérent. Mais cette approche est coûteuse en espace mémoire et en temps de calcul. Mattern [Mat93] propose un algorithme réparti qui détecte un état global cohérent avant de déterminer l'approximation de *GVT*. Cet algorithme utilise la notion de *coupure temporelle* pour distinguer les messages d'événement "avant" et "après" l'état global cohérent.

10.2.1 Inconvénients des approches existantes

Rappelons ci-après les inconvénients des approches existantes pour déterminer *GVT*.

- la présence d'un organe de contrôle central;
- la présence d'un réseau supplémentaire de contrôle ajouté à celui du modèle simulé;
- l'obligation de bloquer tous les *PLs* du système au cours du traitement pour déterminer un état global cohérent.

10.2.2 Motivation

Notre but est de corriger les trois inconvénients ci-dessus. Il s'agit de déterminer un algorithme ayant une fonctionnalité équivalente à celle basée sur le *GVT*, mais exempte de ces trois défauts.

On transforme un *PL* optimiste en un *PL optimiste adaptatif* capable:

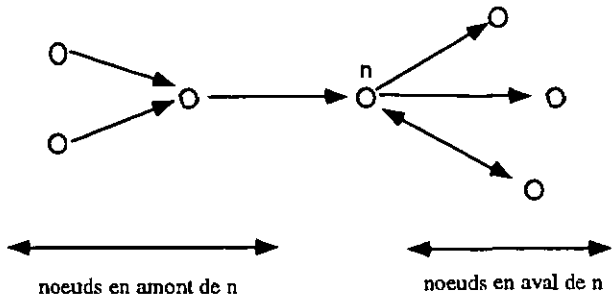


Figure 10.1: Nœud en aval ou en amont de n

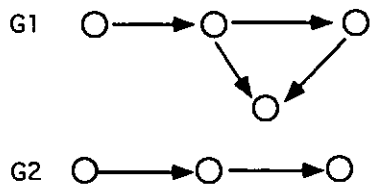


Figure 10.2: Deux graphes sans circuits G1 et G2 indépendants

- de raisonner à partir de son état courant et de son histoire pour déterminer son LT sans bloquer le système entier; ce faisant, nous éliminons également l'organe central de contrôle;
- d'évaluer son état courant pour limiter les risques contractés et pour réduire le coût des rollbacks: la *coopération positive*;
- de corriger entre deux dates locales sûres (le cycle de LT), sa qualité de synchronisation avec ses voisins.

Seul le premier point est traité dans ce chapitre où nous proposons un algorithme ayant la même fonctionnalité que GVT par l'algorithme LT pour déterminer la date locale sûre d'un PL.

10.3 Que désigne-t-on par LT?

Définition 8 (nœud en amont)
Soit $G = (S, A)$ un graphe orienté sans circuit ou GOSC, S l'ensemble des nœuds du graphe et A l'ensemble des arcs. Un nœud $n1$ est en amont d'un nœud $n2$ si et seulement s'il existe un chemin reliant $n1$ à $n2$.

Définition 9 (nœud en aval)
Soit $G = (S, A)$ un GOSC; S l'ensemble des nœuds du graphe et A l'ensemble des arcs. Un nœud $n1$ est en aval d'un $n2$ si et seulement si $n2$ est en amont de $n1$.

Exemple 4
Soit G le graphe réduit (obtenu en contractant ses composantes fortement connexes) d'un modèle simulé et soient $G1$ et $G2$ deux sous-graphes indépendants (figure 10.2). S'il n'existe aucun chemin qui relie un nœud de $G1$ avec un nœud de $G2$ ou inversement, il est

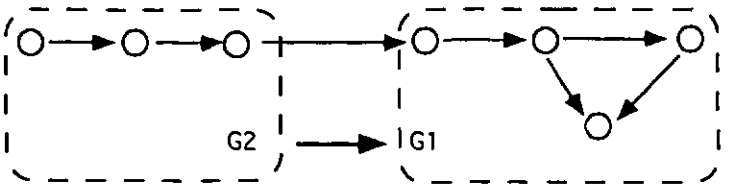


Figure 10.3: Deux graphes G1 et G2 dépendants

évident que les dates virtuelles globales GVT1 et GVT 2 de G1 et G2 peuvent se déterminer de manière indépendante. Par conséquent, il n'est pas nécesssaire de synchroniser un nœud de G1 avec un nœud de G2 afin de déterminer le GVT du graphe simulé G. Par contre, dans le second cas (figure 10.3), il est obligatoire que les nœuds de G1 attendent d'abord le traitement de ceux de G2.

C'est pour éviter le premier type de synchronisation que nous introduisons la notion de *LT* d'un *PL*.

Nous partons du principe que la date locale sûre (l'équivalent de GVT) peut se calculer partiellement depuis les *PLs* sources vers les *PLs* sorties. Chaque *PL* détermine son état local cohérent et le propage aux autres nœuds en aval du graphe. Ainsi, nous définissons un algorithme où les états cohérents de chaque *PL* sont déterminés à partir des canaux et *PLs* en amont de celui-ci.

Définition 10 (ensemble de dépendance)
Soit G le GOSC du modèle simulé où les nœuds sont les PLs. L'ensemble de dépendance d'un PL est l'ensemble des processus logiques dont PL peut recevoir transitivement un message. Il se rapporte à la réception (ou l'attente) transitive des messages par PL. Ce sont les PLs en amont de PL

$$ED(PL) = \{PL_i,\ PL_i \text{ en amont de } PL\}$$

Définition 11 (date locale sûre d'un *PL*)
De manière analogue à la date virtuelle globale $GVT(t)$, nous définissons la date locale sûre $LT(t)$ d'un PL à un instant donné t comme la date à partir de laquelle on ne recevra plus de message d'estampille antérieure provoquant une erreur de causalité. En conséquence, plus aucun rollback ne se fera avant cette date. Une action effectuée à un temps simulé antérieur à $LT(t)$ ne sera alors plus défaite.

Conceptuellement, cette valeur est obtenue en prenant le cliché instantané des PLs de l'ensemble de dépendance. Elle représente la date minimale de tous les messages en attente par le PL.

Calculer la valeur instantanée exacte $LT(t)$ d'un processus étant impossible dans un environnement réparti, une approximation est donc nécessaire pour évaluer une borne inférieure ou égale à cette date. Nous désignerons par *LT* cette borne inférieure estimée.

Importance de LT

La valeur de LT permet de valider les opérations qu'auraient validées le calcul de GVT, notamment:

- la détection de la date à laquelle la simulation est localement valide;
- la récupération de la place mémoire occupée par les états ne pouvant plus être atteints par un retour en arrière: la *récupération des fossiles ("fossil collection")*;
- la validation locale du passé de la simulation et l'exécution des opérations définitives appartenant au passé du PL, par exemple les opérations E/S et le report partiel des calculs statistiques.

Que faut-il pour déterminer LT?

Les difficultés pour déterminer LT proviennent du fait que la date virtuelle locale et les messages non-traités dans la liste des messages à traiter ne suffisent pas pour l'estimer. Le message d'événement recherché peut être momentanément caché dans un canal de communication ou pas encore émis. C'est la raison pour laquelle le calcul de LT par un PL est précédé d'une attente de jeton sur chaque canal d'entrée du PL à laquelle s'adjoint l'observation de l'état local interne du PL. Le rôle de l'attente de jeton sur les canaux d'un PL est de rapporter au PL la plus petite date de son ensemble de dépendance $ED(PL)$.

10.4 Modèle asynchrone de simulation

Un système réparti est constitué d'un ensemble fini de processeurs qui ne communiquent que par envoi de messages. La structure d'un tel système peut être modélisée par un graphe orienté: les sommets de ce graphe sont les processeurs et les arcs représentent les canaux de communication reliant deux processeurs entre eux. A priori, aucune hypothèse particulière n'est faite sur le graphe quant à sa topologie. Les processeurs ne partagent ni mémoire commune ni horloge globale, et aucune hypothèse n'est faite sur leur vitesse d'exécution relative. L'envoi et la réception de messages sont effectués de manière asynchrone mais aucun délai de transfert des messages n'est supposé donné.

La simulation répartie affecte un ou plusieurs PLs à chaque processeur pour simuler un modèle (système physique). Les PLs répartis sur les processeurs forment également un graphe fini.

La vie d'un PL peut être considérée comme une succession d'états. L'action atomique qui modifie l'un de ses états est identifiée à un événement. Un événement est spécifié par deux composantes: l'*action simulée* liée au changement et la *date* du changement. Un calcul local (opération locale) d'un processus est une suite d'événements affectant uniquement son état local, y compris ses canaux d'entrée et de sortie.

10.4.1 Hypothèses

Le système étant représenté par un ensemble de *PL*s, on peut supposer qu'à chaque *PL* est affecté un processeur. Un *PL* peut donc être confondu avec un processeur sans que cela prête à confusion. Les *PL*s sont connectés par un réseau de lignes unidirectionnelles ou canaux.

Un *PL* et un *PL adaptatif* se différencient de la manière suivante: le premier a la liberté de communiquer avec tous les autres *PL*s, alors que le second communique uniquement avec ses proches voisins (auxquels il est relié par des canaux).

Nous passons en revue les hypothèses du modèle simulé.

H1: *Hypothèse de contrôle*

Le réseau sur lequel est construit le système est modélisé par un graphe orienté tel que tout *PL* soit relié par un chemin à un *PL* source et à un *PL* sortie. Ainsi, lorsque les sources lancent le calcul de la date locale sûre ou de terminaison, on a la certitude que l'algorithme diffusant [Ray87] visite tous les *PL*s du réseau.

H2: *Événements engendrés*

La simulation d'un événement courant peut engendrer de nouveaux événements. Ces derniers sont simulés par le même *PL* ou par un *PL* voisin selon l'application de la simulation. Dans le premier cas, ils rejoignent les messages d'*événement internes* et, dans le second cas, ils sont envoyés aux PLs destinataires sous forme de messages d'*événement externes*.

H3: *Hypathèse de réalisabilité ou relation de causalité du système*

Nous supposons que le système est *causal*. Le traitement d'un événement à la date t crée des événements dont l'estampille est ultérieure ou égale à t. Cela provient simplement de la condition de réalisabilité du système simulé.

H4: *Hypathése sur le canal*

Le temps de transmission d'un message à travers un canal unidirectionnel est arbitraire mais fini. Les canaux de communication sont *fiables*: les messages émis ne sont ni perdus ni altérés ni dupliqués.

H5: *Hypathèses du modèle optimiste*

Chaque *PL* contient les variables locales qu'on retrouve dans une simulation optimiste. En particulier:

- une liste triée par rapport à la date de traitement des messages en entrée; une liste triée de messages à la sortie et une liste des états sauvés;
- la date virtuelle locale du *PL*; à un instant particulier, le *PL* simule les événements de sa liste en entrée dont la date de traitement correspond à cet instant. Nous supposons que la durée de la simulation d'un événement est nulle, et que le traitement d'un événement est une action atomique.

H6: *Hypothèses dues à l'algorithme de LT*

- Chaque processus logique (PL) contient la liste de ses canaux d'entrée pour déterminer sa date locale sûre LT . Avant de déterminer LT, le PL attend (au moins) un jeton de chaque canal.
- Un processus (PL) est prêt pour calculer sa date locale sûre suivante s'il a:
  1. reçu de chaque canal d'entrée un jeton au moins;
  2. déterminé la nouvelle valeur du jeton (couleur, temps local sûr).

La dernière hypothèse $H7$ sert à confiner la nouvelle valeur du jeton entre le plus récent événement créé et ceux qui sont déjà envoyés sur les canaux de sortie.

H7: *Hypothèse d'un PL*

A un instant donné, un PL n'a qu'une connaissance partielle des autres PL. Il communique uniquement avec ses voisins, successeurs ou prédécesseurs dans le réseau.

10.4.2 Solution conceptuelle

Elle utilise des jetons (messages de contrôle) pour déterminer l'estimation de la date locale sûre d'un processus logique. Un jeton de date $aTime$ envoyé du PL_{src} au PL_{dest} s'interprète comme suit: le PL_{src} a déterminé une date locale sûre et simulé de manière irrévocable ses activités internes jusqu'à l'instant $aTime$. Ainsi, malgré les rollbacks en amont du PL_{dest}, il ne recevra plus de PL_{src} un jeton dont la date est inférieure à $aTime$. La circulation du jeton préserve aussi la contrainte de causalité car elle satisfait les deux règles suivantes:

- *Règle d'attente du jeton à la réception*

  Chaque PL attend jusqu'à ce qu'il ait au moins un jeton sur chaque canal d'entrée avant de déterminer sa date locale sûre LT. Le traitement d'un jeton entraîne la mise à jour de la date locale sûre et de toutes les opérations qui en résultent.

- *Règle de diffusion du jeton à l'expédition*

  Avant de diffuser la valeur (date locale sûre) du jeton sur ses canaux de sortie, un PL attend jusqu'à ce qu'il soit sûr qu'aucune information ultérieure ne mette en cause cette nouvelle valeur estimée. Cette règle garantit que sur le canal de sortie d'un PL, deux jetons consécutifs portent deux dates sûres non-décroissantes.

  Notons que la cette règle de sortie n'est pas identique à celle du mécanisme de synchronisation sûre où on envoie de messages d'événement ou messages nuls uniquement sur une partie sélectionnée des canaux de sortie.

Le traitement du jeton par un PL se résume à:

1. attendre un jeton sur chaque canal d'entrée;

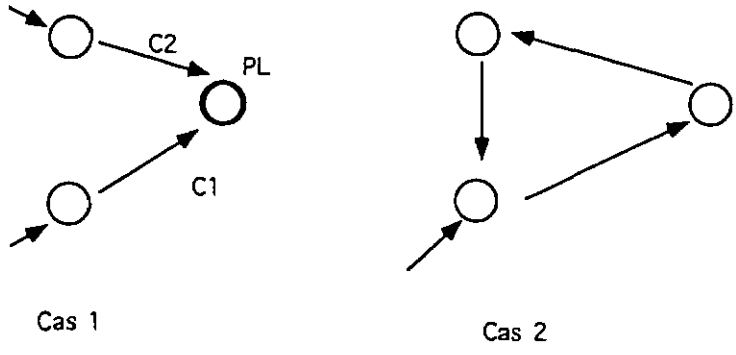


Figure 10.4: Situations typiques d'impasse

2. déterminer la nouvelle valeur de LT;

3. faire la mise à jour du jeton: sa couleur qui détermine la phase intermédiaire de calcul à l'intérieur ou l'extérieur d'une composante fortement connexe et son temps en lui affectant LT;

4. envoyer le jeton sur chaque canal de sortie.

Remarque 12 (Observation instantanée)
Dans un simulateur séquentiel, l'heure observée sur l'horloge globale représente la date sûre pour l'ensemble de la simulation. Ce n'est pas le cas dans une simulation répartie puisque chaque PL gère individuellement sa date locale sûre.

La date globale du simulateur H (l'équivalent du GVT classique) est la plus petite date de toutes les dates sûres locales LTs des PLs et des estampilles des messages en transit. Ainsi, on peut affirmer que le simulateur réparti a simulé définitivement le modèle (système physique) jusqu'à la date H, même si certains PLs ont simulé leur processus physique au-delà de cette date.

10.4.3 Absence d'impasse

On peut se demander si cet algorithme pour déterminer la date locale sûre ne mène pas à une situation d'impasse, dans la mesure où il ressemble à l'approche sûre qui possède cet inconvénient. Rappelons les deux situations typiques (cf. figure 10.4) qui mènent à une impasse dans une simulation sûre.

Absence de famine sur un canal d'entrée

Peut-on avoir un PL qui attend indéfiniment un jeton d'un de ses canaux d'entrée (cf. figure 10.4 cas 1)? La réponse est négative, par la règle de diffusion des jetons sur les canaux de sortie d'un PL. En utilisant l'algorithme diffusant [Ray87], chaque PL a au moins un jeton sur son canal d'entrée. Cette absence de famine est due à la modification de la règle de sortie des jetons en se rapportant à la règle de sortie dans le mécanisme de synchronisation sûre. Rappelons le: dans l'approche sûre on envoyait des messages d'événement sur des canaux de sortie sélectionnés et la famine provenait des canaux

qui n'étaient pas alimentés. Mais dans notre approche, la règle de la diffusion (plus sporadique) de jetons envoie au moins un jeton sur les canaux de sortie de chaque PL.

Absence de circuits

Pour éviter les impasses dues à un circuit (la figure 10.4 cas 2), il faut considérer le graphe (sans circuit) obtenu en contractant les composantes fortement connexes (cf. section 10.3). Cela se fait par un traitement préalable qui détermine les composantes fortement connexes du graphe, et réduisent le graphe en contractant celles-ci. Ainsi, on détecte les circuits, puis par un traitement particulier des composantes connexes, on veille à ce qu'un jeton ne tourne pas indéfiniment dans un circuit. C'est d'ailleurs l'intérêt du traitement préalable de l'algorithme proposé.

10.5 Fonctionnement de l'algorithme LT

Nous distinguons deux types de messages qui circulent dans le système: les messages d'événement du modèle simulé et les messages de contrôle (jetons). Nous nous concentrons sur le traitement des messages de contrôle pour évaluer la valeur de LT d'un PL.

Soit n L'ensemble des PLs du système. L'algorithme LT comporte deux phases:

- une phase de pré-traitement où les n PLs sont partitionnés en composantes fortement connexes. L'objectif de ce traitement préalable est de déterminer le graphe acyclique associé au graphe simulé, en contractant les composantes fortement connexes;
- une phase de traitement où la valeur de LT est déterminée conjointement au traitement des messages d'événement de la simulation optimiste.

Définition 12
L'ensemble des ***canaux d'entrée*** *pour un PL faisant partie d'une composante connexe est l'ensemble des canaux d'entrée de ce PL provenant des PLs extérieurs à la composante.*

L'ensemble des ***canaux de sortie*** *pour un PL faisant partie d'une composante fortement connexe est l'ensemble de ses canaux de sortie aboutissant à un PL extérieur à la composante.*

10.5.1 Phase de pré-traitement

Elle sert à:

1. identifier les composantes fortement connexes dans le système;
2. déterminer les canaux d'entrée et de sortie de chaque PL de la composante fortement connexe (cf. la définition ci-dessus) ;
3. fixer pour chaque composante fortement connexe un élément représentatif que nous désignons par son *élément de tête*;

4. identifier un *arbre de recouvrement* de la composante fortement connexe dont la racine est l'élément de tête. Cet arbre contient tous les nœuds (PLs) de la composante fortement connexe. En effet, il sert à calculer la valeur estimée de LT dans une composante fortement connexe;

5. identifier, pour chaque nœud feuille (PL) de l'arbre de recouvrement un chemin élémentaire pour se rendre à la racine (élément de tête) de la composante fortement connexe;

6. mémoriser dans la racine (l'élément de tête) tous les nœuds feuilles (PLs) de la composante fortement connexe. La valeurs estimée LT est le minimum des valeurs retournées par ces derniers;

Concrètement, chaque PL d'une composante contient pour calculer LT les pointeurs:

- *goToHead* qui pointe au nœud (PL) sur le chemin du retour à l'élément de tête à partir de chaque feuille de l'arbre;

- *successors* qui pointe sur les nœuds–fils (PL) de l'arbre de recouvrement de la composante; les nœuds feuilles de l'arbre de recouvrement ont leur *successors* à nil.

et deux listes:

- la liste de ses canaux d'entrée, ce sont les canaux d'accès à la composante par le PL;

- la liste de ses canaux de sortie, ce sont les canaux d'accès à d'autres composantes par le PL;

10.5.2 Phase de traitement

Conceptuellement, l'algorithme utilise un jeton pouvant prendre quatre *couleurs:* rouge, vert, bleu ou jaune (figure 10.5). Un jeton de couleur vert, bleu ou jaune circule à l'intérieur d'une composante, alors que le jeton rouge propage la valeur estimée de LT d'une composante à une autre. On associe à une couleur l'action à exécuter à la réception du jeton.

L'algorithme fonctionne en quatre étapes (cf. figure 10.5):

1. Aller à l'élément de tête de la composante pour lancer le calcul de la nouvelle valeur de LT ("ExecuteGotoHead" dans la figure 10.5); pendant cette étape, le jeton est de couleur *rouge* (cf. figure 10.6).

2. Lancer le calcul de la nouvelle valeur de LT ("ExecuteCompute"). L'estimation se fait en suivant l'arbre de recouvrement de la composante fortement connexe. Chaque PL à la réception du jeton calcule le minimum sur son chemin et le propage à ses nœuds–fils. Pendant cette étape le jeton est de couleur *verte* (cf. figure 10.7).

3. Rejoindre l'élément de tête de la composante ("ExecuteGoBackToHead"); pendant cette étape, le jeton est de couleur *bleue* (cf. figure 10.8).

   En rejoignant la racine de l'arbre de recouvrement, les feuilles rapportent leur valeur partielle LT (jeton). La racine de l'arbre détermine le minimum de ces dernières pour obtenir la nouvelle valeur estimée de LT.

   Si la nouvelle valeur estimée est différente de la précédente, alors exécuter l'étape 4, sinon sortir de l'algorithme. Il ne sert à rien de propager plus loin la valeur précédente.

4. Propager la nouvelle valeur estimée de LT aux composantes fortement connexes voisines en visitant à nouveau chaque PL de la composante ("ExecutePropagate"). La propagation sur tous les canaux de sortie de la composante se fait en empruntant l'arbre de recouvrement de la composante. Pendant cette étape, la couleur du jeton est *jaune* (cf. figure 10.9).

Concrètement, quelles sont les actions que doit exécuter un (PL) à la réception d'un message de contrôle (jeton)?

Un jeton est un message de contrôle ayant la structure suivante:

jeton ::= <couleur, LT, expéditeur, destinataire> où

couleur: action à exécuter;

LT: valeur de la date locale sûre du PL expéditeur;

expéditeur: PL ou nœud feuille expéditeur du jeton;

destinataire: PL destinataire du jeton.

Voici la fonctionnalité des procédures utilisées dans la description des algorithmes:

PurgeMemoryUntil: aTime

collectionne les fossiles jusqu'au temps *aTime*.

BroadcastOutSide: jeton(red, LT)

envoie le *jeton(red, LT)* à tous les canaux de sortie du PL aboutissant à d'autres composantes fortement connexes.

SendToken:: jeton(color, aTime) to: receiver

envoie le jeton *jeton(color, aTime)* au receveur: *receiver*.

Remarque 13

1. Dans les algorithmes proposés, la fonction $LTnew()$ est calculée d'après la fonction suivante:

   $Minimum(jeton.aTime, localToken, internalClock)$

   qui retourne le minimum des valeurs d'entrée; *localToken* désigne la plus petite estampille des jetons sur les canaux d'entrée provenant de l'extérieur de la composante fortement connexe au PL;

   internalClock désigne le minimum entre la plus petite estampille des messages non-traités dans la liste d'entrée du PL et son LVT;

2. On attend toujours un jeton de couleur rouge sur les canaux d'entrée d'une composante fortement connexe restreinte à un PL.

3. Il y a implicitement une *faible synchronisation asynchrone* des jetons à l'intérieur d'une composante fortement connexe pour garantir à la sortie d'une composante une valeur cohérente LT du jeton. Sans quoi, on risque de propager une valeur incohérente de LT aux composantes. Elle s'exprime par la mise à jour des jetons sur chaque canal d'entrée de la composante avant d'évaluer LT.

4. Peut-on confondre l'élément de tête avec les autres PLs? La réponse est négative car il garantit qu'on parcourra tous les PLs d'une composante à partir de lui en évitant les boucles du jeton dans une composante fortement connexe. Il désigne la racine de l'arbre de recouvrement d'une composante fortement connexe.

10.6 Formulation de l'avancement de LT

Soit $T(t)$ l'ensemble des messages momentanément en attente de traitement (en transit ou dans la liste des messages en entrée) du PL. Nous formulons l'invariant (sûreté) et la condition d'avancement de la date locale sûre de la manière suivante:

- *Invariant ("condition")*

  La date locale sûre d'un processus logique PL n'avance jamais au-delà de l'estampille des messages d'événement en attente de traitement par PL. Soit $T_{PL}(t)$ l'ensemble des messages momentanément en attente de traitement (en transit ou dans la liste des messages d'entrée) du PL, en conséquence en $\forall t$ et pour tout PL:

$$\forall m \in T_{PL}(t) \quad LT_{PL}(t)(t) \leq ts(m)$$

- *Condition de vivacité ("progess requirement")*

  La *condition de vivacité* affirme, pour chaque PL, il existe toujours au moins un message en attente de traitement dont le traitement augmentera la date sûre locale du PL. Cela veut dire que pour tout PL,

$$\exists m \in T_{PL}(t), \textit{tel que après le traitement de } m \quad LT_{PL}(t) \geq ts(m)$$

  Remarque 14
  Une fois qu'un message m en attente de traitement par PL, $m \in T(t)$ est traité, il n'est plus en attente de traitement par celui-ci, par conséquent $m \notin T(t)$; et l'*invariant* démeure toujours vrai.

Pour prouver l'exactitude de l'algorithme à savoir l'invariant et la vivacité de l'algorithme LT, nous allons utiliser les mêmes techniques de preuve formelle que d'autres chercheurs [Tel91], [BM81], [XGUC95]. C'est le but du prochain chapitre.

10.7 Algorithmes de traitement des jetons

```
ReceiveToken::jeton(color, aTime)
begin
  CASE color OF
    |red:    ExecuteGotoHead:: jeton(rouge, aTime);
    |green:  ExecuteCompute:: jeton(vert, aTime);
    |blue:   ExecuteGoBackToHead:: jeton(bleu, aTime);
    |yellow: ExecutePropagate:: jeton(jaune, aTime);
  END of CASE
end
```

Figure 10.5: *Traitement associé à la réception d'un jeton*

```
ExecuteGotoHead:: jeton(rouge, aTime);
begin
  IF sender.goToHead = nil THEN
    Check jeton on each PL input component channels
    IF sender.successors <> nil THEN
      FOR each nodeSon ∈ successors DO
        SendToken:: jeton(vert, LTnew()) to: sender.nodeSon;
      END FOR
    ELSE
      BroadcastOutSide:: jeton(rouge, LTnew());
    END IF;
  ELSE
    SendToken:: jeton(rouge, LT) to: sender.goToHead;
  END IF
end.
```

Figure 10.6: *Traitement du jeton rouge (se rendre à l'élément de tête)*

```
ExecuteCompute:: jeton(vert, aTime);
begin
  Check jeton on each PL input component channels
  IF (sender.successors <> nil) THEN
    FOR each nodeSon ∈ successors DO
      SendToken:: jeton(vert, LTnew()) to: sender.nodeSon;
    END FOR
  ELSE
    LT := LTnew();
    IF (previousLT ≠ LT) THEN
      SendToken:: jeton(bleu, LT) to: sender.goToHead;
    END IF
  END IF
end.
```

Figure 10.7: *Traitement du jeton vert (calcul de LT)*

```
ExecuteGoBackToHead:: jeton(bleu, LT);
begin
  IF sender.goToHead = nil THEN
    PurgeMemoryUntil: jeton.LT;
    BroadcastOutSide: jeton(rouge, LT);
    IF sender.successors ≠ nil THEN
      FOR each nodeSon ∈ successors DO
        SendToken:: jeton(yellow, LT) to: sender.nodeSon;
      END FOR
    END IF;
  ELSE
    SendToken:: jeton(blue, LT) to: sender.goToHead;
  END IF
end.
```

Figure 10.8: *Traitement du jeton bleu (rapporter LT à l'élément de tête)*

```
ExecutePropagate:: jeton(jaune, LT);
begin
  PurgeMemoryUntil: jeton.LT;
  BroadcastOutSide: jeton(rouge, LT);
  IF sender.successors ≠ nil THEN
    FOR each nodeSon ∈ successors DO
      SendToken:: jeton(yellow, LT) to: sender.nodeSon;
    END FOR
  END IF;
end.
```

Figure 10.9: *Traitement du jeton jaune (propagation de LT)*

10.8 Originalité de notre approche

L'algorithme pour déterminer LT est une synthèse des deux approches classiques: sûre et optimiste. Il se distingue des algorithmes existants en autorisant à la fois une faible synchronisation des jetons (mise à jour des jetons sur chaque canal d'entrée) et la liberté de traiter les messages d'événement du modèle simulé de manière optimiste.

Elle se distingue de toutes les approches existantes par sa manière de déterminer un état local cohérent pour chaque processus logique, alors que les approches recherchent un état global cohérent. Nous évitons l'exigence d'un état global cohérent en observant uniquement, pour un PL, les états locaux cohérents de son ensemble de dépendance $ED(PL)$. ***L'avantage de notre approche réside dans le fait que chaque PL dispose d'une date locale sûre à partir de laquelle on ne recevra plus de message d'estampille antérieure. Cette date permet de valider exactement les mêmes opérations qu'aurait validées GVT.***

Il existe des algorithmes de GVT qui déterminent des valeurs intermédiaires $Lgvt$ [DFW94] qui interviennent dans le calcul de GVT. Notre algorithme se distingue de ceux-là en offrant une date locale sûre et non une valeur intermédiaire.

10.9 Conclusion

Au lieu de calculer d'abord un état global cohérent pour déterminer GVT, nous avons introduit la notion d'ensemble de dépendance d'un PL pour déterminer sa date locale sûre LT. La date locale sûre d'un PL correspond intuitivement au GVT du graphe simulé restreint aux PLs de son ensemble de dépendance $ED(PL)$. Pour un modèle simulé, les LT sont calculés des PLs sources aux PLs de sortie.

Le modèle de calcul de LT répond bien aux objectifs fixés en corrigeant les défauts majeurs des algorithmes de GVT. Il permet en outre de:

- déterminer l'équivalent de GVT sans figer ou geler les PLs;

- déterminer la date locale sûre sans avoir à examiner tous les PLs qui se trouvent dans le système; on élimine ainsi la présence d'un organe central de contrôle;
- calculer sans le recours à un réseau supplémentaire, ce que nécessite le calcul de GVT.

De plus, le modèle proposé est assez robuste car il s'applique également aux réseaux sous-jacents comportant des circuits. Ce modèle de calcul de LT permet:

- de déterminer la valeur de LT conformément à l'évolution du système réel dans le temps;
- à chaque PL de calculer sa date locale sûre à partir de son ensemble de dépendance. Seuls les PLs qui sont en relation d'échange d'information sont pris en considération dans le modèle de calcul. Le modèle de calcul de LT est alors beaucoup plus souple que celui de GVT;
- à chaque PL de s'adapter particulièrement à la gestion des opérations d'entrée/sortie de la simulation interactive répartie [Fis95, pp359-369].

Chapitre 11

L'exactitude de l'algorithme LT

Résumé

Ce chapitre propose une approche formelle pour définir la date locale sûre d'un PL ainsi que sa valeur estimée LT. Nous utilisons les mêmes techniques de démonstration formelle que Ted [Tel91], Boyer et al. [BM81], D'souza [DFW94], Xiao et al. [XGUC95] pour démontrer l'*exactitude de l'algorithme de LT*. La technique de preuve comporte deux parties: la preuve de l'exactitude partielle et la preuve de l'exactitude totale. La première partie prouve que LT est une fonction monotone non décroissante ne dépassant jamais la valeur exacte ELT, alors que la seconde partie prouve que la valeur LT, d'après l'algorithme, croît de manière monotone lorsque sa valeur exacte ELT augmente.

11.1 Notations des messages à traiter

Nous donnons quelques définitions et notations relatives aux messages à traiter: *message en attente de traitement*. Pour un processus logique PL_k donné, il existe deux sortes de messages en attente de traitement:

- les messages non-traités dans la liste d'entrée du PL_k;
- les messages non-traités en transit dans un canal (PL_i, PL_k) qui sont déjà envoyés par PL_i mais qui ne sont pas encore parvenus au destinataire PL_k.

Soit P: l'ensemble des PLs dans le système; $ED^k = ED(PL_k)$: l'ensemble de dépendance du PL_k; un $PL_j \in ED^k$ est noté PL_j^k.

11.1.1 Messages non-traités dans un PL

Soient: IQ_j^k: l'ensemble des messages dans la liste d'entrée du PL_j^k. Cette liste est composée de deux sous-listes disjointes: UQ_j^k (pour "unprocessed input queue") ou l'ensemble des messages non-traités dans IQ_j^k et PQ_j^k l'ensemble des messages déjà traités par le PL_j^k; ainsi pour un PL_j^k donné $UQ_j^k \cup PQ_j^k = IQ_j^k$ et $UQ_j^k \cap PQ_j^k = \emptyset$.

$UQ_j^k(t)$: l'ensemble des messages de la liste d'entrée ED^k non-traités dans PL_j^k à l'instant réel t;

$UQ^k(t)$: l'ensemble des messages des listes d'entrée non-traités de ED^k à l'instant réel t; alors

$$UQ^k(t) = \bigcup_{j:\forall PL_j \in ED^k} UQ_j^k(t)$$

11.1.2 Messages non-traités en transit

Les *messages en transit* sont des messages envoyés d'un PL_{src} source à un autre PL_{dest} destinataire mais qui ne sont pas encore parvenus au PL_{dest}. Ces messages sont momentanément dans un canal de communication. Désignons par:

$TR_{i,j}^k(t)$: l'ensemble des messages de ED^k en transit du PL_i^k vers le PL_j^k à l'instant réel t;

$TR_j^k(t)$: l'ensemble des messages de ED^k en transit dans les canaux d'entrée du PL_j^k à l'instant réel t;

$$TR_j^k(t) = \bigcup_{\forall i:PL_i \in ED^k} TR_{i,j}^k(t)$$

$TR^k(t)$: l'ensemble des messages de ED^k en transit à l'instant réel t;

$$TR^k(t) = \bigcup_{\forall j:PL_j \in ED^k} TR_j^k(t)$$

11.1.3 Messages en attente de traitement

Pour un PL, ses *messages en attente* regroupent l'ensemble de ses messages non-traités dans son UQ et l'ensemble de ses messages en transit dans ses canaux d'entrée. Désignons par:

$T_{i,j}^k(t)$: l'ensemble des messages de ED^k en attente de traitement du PL_i^k vers le PL_j à l'instant réel t;

$T_j^k(t)$: l'ensemble des messages de ED^k en traitement vers le PL_j à l'instant réel t;

$$T_j^k(t) = \bigcup_{\forall i:PL_i \in ED^k} T_{i,j}^k(t)$$

$T^k(t)$: l'ensemble des messages de ED^k en attente de traitement à l'instant réel t.

$$T^k(t) = \bigcup_{\forall j:PL_j \in ED^k} T_j^k(t)$$

Remarque:

$$T_j^k(t) = TR_j^k(t) \cup UQ_j^k(t)$$

et

$$T^k(t) = TR^k(t) \cup UQ^k(t)$$

11.1.4 Date locale sûre

La date locale sûre d'un PL_k ELT_k ("Exact Local Time") est une valeur instantanée représentant la progression locale de la simulation au sein du PL_k. Toute action entreprise avant cette date est irrévocable, hors de la portée d'un éventuel rollback. Elle représente le minimum entre (i) la plus petite date virtuelle locale des PLs de $ED(PL_k) = ED^k$ et (ii) la plus petite estampille des messages en attente de traitement par PL_k. Définissons formellement cette valeur relative à un PL_k.

$lvt_j^k(t)$: la date virtuelle locale du PL_j^k à l'instant réel t.

$T^k(t)$: l'ensemble des messages de ED^k en attente de traitement à l'instant réel t.

ELM_j^k:le minimum local de la date exacte du PL_j^k à l'instant réel t. Sous forme d'équation, il est défini par:

$$ELM_j^k(t) = \min\left\{lvt_j^k(t), \min\{ts(m), m \in T_j^k(t)\}\right\}$$

Nous définissons la valeur instantanée ou exacte de la date locale sûre ELT_k par:

$$ELT_k(t) = \min_{\forall j: PL_j \in ED^k}\{ELM_j^k\}$$

en développant ELM_j^k on a:

$$ELT_k(t) = \min\left\{\min\{lvt_j^k(t), \forall j : PL_j \in ED^k\}, \min\{ts(m), m \in T^k(t)\}\right\}$$

Remarque

La définition exacte de la date locale sûre du PL_k à un instant t se rapporte à son ensemble de dépendance: ED^k. C'est le minimum instantané des ELM_j^k de chaque PL_j^k de son ensemble de dépendance.

11.1.5 Estimation de la date locale sûre

Pour un PL_j de ED^k, définissons les paramètres qui interviennent dans l'estimation de sa date locale sûre interne. Soient:

$lvt_j^k(t)$: la date virtuelle locale du PL_j^k à l'instant réel t;

$UQ_j^k(t)$: la liste des messages non-traités dans PL_j^k à l'instant réel t;

$token_{i,j}$: la date du dernier jeton reçu le long du canal d'entrée (PL_i^k, PL_j^k);

Alors la valeur estimée $LT_j^k(t)$ du PL_j^k se définit par:

$$LT_j^k(t) = \min\left\{lvt_j^k(t), \min_{m \in UQ_j^k(t)}\{ts(m)\}, \min_{\forall l: PL_l \in ED^k}\{token_{l,j}(t)\}\right\}$$

Remarques

1. Il est clair qu'un message de contrôle ou *jeton* n'a pas son correspondant dans le système physique. Le rôle du jeton est d'amener à un PL le minimum des lvts et des messages en attente de traitement de tous les PLs qui étaient sur son chemin. Ainsi, il ramène à un PL le minimum de la date locale sûre des PLs sur son chemin depuis la source. Autrement dit, les jetons des canaux d'entrée servent à déterminer l'estimation du minimum local des PLs en amont du PL courant qui traite le jeton.

2. Pour des raisons de clarté, nous abrégeons la notation $LT_j^k(t)$ en $LT_j(t)$ où l'exposant k relatif à ED^k est ignoré. De plus, notre raisonnement se rapporte à ED^k. La $i^{ième}$ estimation de la date sûre locale $LT_j(t_i)$ à l'instant $t = t_i$ est notée $LT_j^{(i)}$. On a:

$$LT_j^k(t_i) = \min\left\{lvt_j^k(t_i), \min_{m \in UQ_j^k(t_i)}\{ts(m)\}, \min_{\forall l: PL_l \in ED^k}\{token_{l,j}(t_i)\}\right\}$$

et $LT_j^{(i)} = LT_j(t_i)$.

11.2 Exactitude partielle

Intuitivement, l'exactitude partielle d'un algorithme signifie que l'algorithme ne fait rien de faux. Elle se rapporte aux propriétés de *sûreté* ou ("safety property"). Formellement, elle est formulée en terme d'*invariant*, une condition toujours vraie au cours de l'exécution de l'algorithme. Pour notre algorithme l'invariant est le suivant:

$$\forall PL_j^k \qquad \forall t \qquad LT_j^k(t) \leq ELT_j^k(t)$$

En d'autres termes, *à chaque instant de la simulation, la valeur estimée de la date locale sûre d'un PL ne dépasse jamais sa valeur exacte instantanée.* Ainsi, en faisant abstraction de l'indice et de l'exposant dans la notation, la condition devient:

$$\forall PL \in P, \qquad \forall t \qquad LT(t) \leq ELT(t)$$

11.2.1 Preuve

La preuve se fait par induction complète.

Ancrage

A l'instant initial t_0 de la simulation: pour tout $PL \in P$, $lvt(t_0) = 0$, en substituant lvt dans la définition de l'estimation de LT on a, d'une part:

$$LT_j^k(t_0) = \min \left\{ lvt_j^k(t_0), \min_{m \in UQ_j^k(t_0)} \{ts(m)\}, \min_{\forall l: PL_l \in ED^k} \{token_{l,j}(t_0)\} \right\}$$

et, d'autre part dans la définition de ELT:

$$ELT_k(t_0) = \min \left\{ \min_{\forall j: PL_j \in ED^k} \{lvt_j^k(t_0)\}, \min_{m \in T^k(t_0)} \{ts(m)\} \right\} = 0$$

Ainsi donc: $\forall PL, \quad ELT(t_0) = LT(t_0) = LT^{(0)} = 0$

Pas d'induction

Nous supposons que pour chaque PL, la condition soit toujours vraie jusqu'à l'instant $t = t_i$ ou jusqu'à la $i^{ième}$ estimation de sa date locale sûre. Ensuite, nous cherchons à prouver qu'elle est aussi vraie à la $(i+1)^{ième}$ estimation à la date t_{i+1}.

Convention: pour la clarté de la preuve, nous désignons $LT_j^k(t_i)$ par $LT^{(i)}$ et $ELT_j^k(t_i)$ par $ELT(t_i)$. Tant qu'il n'est pas nécessaire d'exprimer la notation complète, nous utiliserons la notation abrégée. Par défaut, la preuve se rapporte à un PL_j^k.

Par hypothèse:

$$\forall t \leq t_i, \qquad LT^{(i)}(t) \leq ELT(t)$$

en utilisant la définition de $LT^{(i)}$:

$$LT_j^k(t_i) = \min \left\{ lvt_j^k(t_i), \min_{m \in UQ_j^k(t_i)} \{ts(m)\}, \min_{\forall l: PL_l \in ED^k} \{token_{l,j}(t_i)\} \right\}$$

si $LT^{(i)} \leq ELT(t_i) \qquad \forall t \leq t_i$, trois cas peuvent se présenter:

1. $lvt_j^k(t_i) = LT^{(i)}$
2. $\exists m \in UQ_j^k(t_i)$ tel que $ts(m) = LT^{(i)}$ et $LT^{(i)} \leq ELT(t_i)$
3. $\exists l$ tel que $token_{l,j}(t_i) = LT^{(i)}$,
   autrement dit, il existe un canal d'entrée (PL_l, PL_j) duquel provient le jeton qui détermine la valeur de $LT^{(i)}$.

Nous prouvons le pas d'induction par l'absurde. Cela revient à prouver que $LT^{(i+1)}$ ne satisfait pas la condition $LT^{(i+1)} \leq ELT(t_{i+1})$ et de ce fait engendrer une contradiction.

Si $LT^{(i+1)}$ est incorrect, alors une des deux conditions suivantes est vérifiée:

Cas I $LT^{(i+1)} < LT^{(i)}$

Cas II $LT^{(i+1)} > ELT(t_{i+1})$

Notre tâche consiste à prouver que ces deux cas ne peuvent jamais se présenter.

Prouvons que le cas I: $LT^{(i+1)} < LT^{(i)}$ n'apparaît jamais

Selon la définition de $LT^{(i)}$, trois cas doivent être examinés:

Cas I.1: $lvt_j^k(t_i) = LT^{(i)}$

A la $(i+1)^{ième}$ estimation de LT par PL_j^k à l'instant t_{i+1} deux cas peuvent se présenter:

1. soit $lvt_j^k(t_i) = lvt_j^k(t_{i+1})$

   Si $lvt_j^k(t_i) = lvt_j^k(t_{i+1})$ et par hypothèse d'induction $lvt_j^k(t_i) = LT^{(i)}$ alors
   $LT^{(i+1)} = LT^{(i)}$ c.q.f.d

2. soit $lvt_j^k(t_i) \neq lvt_j^k(t_{i+1})$

   à la $i^{ième}$ estimation, il existe un message m tel que $ts(m) = LT^{(i)}$; ce qui signifie qu'à l'instant t_i, il existe un message m en cours de traitement dont $ts(m) = lvt_j^k(t_i) = LT^{(i)}$.

   Mais à la $(i+1)^{ième}$ estimation de LT par PL_j^k à l'instant t_{i+1}, m a déjà fini son traitement et le processus est en train de traiter un autre message $\bar{m}$:

   (a) soit $\bar{m}$ est engendré par m alors $ts(\bar{m}) \geq ts(m)$ par le principe de réalisabilité; mais comme $LT^{(i+1)} = ts(\bar{m})$ et $LT^{(i)} = ts(m)$ on a donc:
   $LT^{(i+1)} \geq LT^{(i)}$ c.q.f.d

   (b) soit $\bar{m} \in UQ_j^k(t_i$, alors $ts(\bar{m}) \geq ts(m)$, sinon $\bar{m}$ aurait été choisi à la place de m à l'instant t_i; mais comme $LT^{(i+1)} = ts(\bar{m})$ et $LT^{(i)} = ts(m)$ on a donc:
   $LT^{(i+1)} \geq LT^{(i)}$ c.q.f.d

Cas I.2: $\exists m \in UQ_j^k(t_i)$ avec $ts(m) = LT^{(i)}$

Soit $m \in UQ_j^k(t_i)$ le message dont le $ts(m) = LT^{(i)}$; à la $(i+1)^{ième}$ estimation de LT par PL_j^k à l'instant t_{i+1}, soit $m \notin UQ_j^k(t_i)$ soit $m \in UQ_j^k(t_i)$.

a. si $m \notin UQ_j^k(t_i)$

   m est en cours de traitement ou aurait été traité pour engendrer éventuellement un message $\bar{m}$; par le principe de réalisabilité $ts(\bar{m}) \geq ts(m)$, il s'ensuit qu'à l'instant t_{i+1} PL_j^k ne possède aucun message $\bar{m}$ dont $ts(\bar{m}) < ts(m)$; on a forcément alors $LT^{(i+1)} \geq ts(m)$ mais comme par hypothèse $ts(m) = LT^{(i)}$ on conclut:
   $LT^{(i+1)} \geq LT^{(i)}$ c.q.f.d

b. si $m \in UQ_j^k(t_i)$, m n'est pas encore traité alors $m \in UQ_j^k(t_{i+1})$, ce qui signifie que $ts(m) = LT^{(i+1)}$. Or par hypothèse $ts(m) = LT^{(i)}$; il en découle que:
$LT^{(i+1)} = LT^{(i)}$ c.q.f.d

Cas I.3: $\exists l$ **tel que** $token_{l,j}(t_i) = LT^{(i)}$

Il existe à l'instant t_i un canal d'entrée (PL_l, PL_j) duquel provient le jeton qui détermine la valeur de $LT^{(i)}$; autrement dit, $LT^{(i)} = token_{l,j}(t_i)$. A l'instant $t_{i+1} > t_i$, deux cas peuvent se présenter:

(i) soit $token_{l,j}(t_i) = token_{l,j}(t_{i+1})$, ce qui signifie exactement:
$LT^{(i+1)} = LT^{(i)}$ c.q.f.d

(ii) soit $token_{l,j}(t_i) \neq token_{l,j}(t_{i+1})$, ce qui signifie que le PL_j^k a recu un autre jeton le long du canal (PL_l, PL_j). Mais comme chaque PL applique la méthode sûre aux jetons (cf. les règles d'attente de réception et d'expédition d'un jeton) on a forcément: $token_{l,j}(t_{i+1}) > token_{l,j}(t_i)$ et ensuite
$\forall m : PL_m \quad token_{m,j}(t_{i+1}) \geq token_{l,j}(t_i)$ donc:

$$\min_{\forall m : PL_m} \{token_{m,j}(t_{i+1})\} \geq token_{l,j}(t_i)$$

autrement dit:
$LT^{(i+1)} \geq LT^{(i)}$ c.q.f.d

Conclusion partielle

$LT^{(i+1)} \geq LT^{(i)} \quad \forall i \geq 0$

Prouvons que Cas II: $LT^{(i+1)} > ELT(t_{i+1})$ **n'apparaît jamais**

Nous distinguons deux cas:

1. soit $LT^{(i+1)} = LT^{(i)}$
2. soit $LT^{(i+1)} \neq LT^{(i)}$

Cas II.1 $LT^{(i+1)} = LT^{(i)}$

Par hypothèse d'induction $LT^{(i)} \leq ELT(t_i)$ on a: $LT^{(i+1)} \leq ELT(t_i)$. Par la définition de $ELT(t)$ et grâce au principe de réalisabilité du modèle simulé, on a pour $t_{i+1} > t_i$, $ELT(t_i) \leq ELT(t_{i+1})$. On déduit alors:
$LT^{(i+1)} \leq ELT(t_{i+1})$ c.q.f.d

Cas II.2 $LT^{(i+1)} \neq LT^{(i)}$ Comme nous l'avons démontré au cas I, on a forcément: $LT^{(i+1)} \geq LT^{(i)}$.

Soit m le message d'événement réalisant le minimum instantané à l'instant t_i, qui signifie $ELT^{(i)} = ts(m)$. A l'instant t_{i+1} deux cas peuvent se présenter:

a. soit le message m a changé dans l'intervalle temporel $[t_i..t_{i+1}]$

b. soit le message m n'a pas changé

Cas II.2.a: m a changé

Le message réalisant le minimum instantané à l'instant t_i a changé de valeur à l'instant t_{i+1}, ce qui signifie par définition de $ELT(t)$ que $ELT(t_{i+1}) \neq ELT(t_i)$. De plus, par principe de réalisabilité du modèle simulé, on a forcément: $ELT(t_{i+1}) > ELT(t_i)$.

Deux cas peuvent alors se présenter à l'instant t_{i+1}:

(i) soit PL_j^k n'a pas perçu la modification; alors la valeur instantanée $ELT(t_{i+1})$ est une modification d'un de ses PLs en amont de son ensemble de dépendance, ainsi pour le PL_j^k on a:

d'une part: $LT^{(i)} < LT^{(i+1)} \leq ELT(t_i)$

et d'autre part: $ELT(t_i) < ELT(t_{i+1})$,

ce qui implique: $LT^{(i+1)} < ELT(t_{i+1})$ c.q.f.d

(ii) soit PL_j^k a perçu la modification; ce qui signifie que le message m appartient à l'état local (interne) de PL_j^k, alors:

$LT^{(i+1)} = ELT(t_{i+1})$ c.q.f.d

Cas II.2.b: m n'a pas changé

Le message m réalisant le minimum instantané à l'instant t_i n'a pas changé à l'instant t_{i+1}; ce qui se traduit par:

$ELT(t_i) = ELT(t_{i+1})$

Et si par hypothèse de travail $LT^{(i)} < LT^{(i+1)}$, alors le changement de l'estimation est dû probablement à la propagation d'une valeur plus proche de $ELT(t_i)$; ainsi donc: $LT^{(i)} \neq LT^{(i+1)} \leq ELT(t_i)$. On déduit des deux paragraphes ci-dessus:

$LT^{(i+1)} \leq ELT(t_{i+1})$ c.q.f.d

Résumons la preuve

Nous venons de prouver par les deux cas I et II, les résulats suivants:

$$\forall i \geq 0 \quad LT^{(i+1)} \geq LT^{(i)}$$

et

$$\forall i \geq 0 \quad LT^{(i+1)} \leq ELT(t_{i+1})$$

ainsi

$$\forall PL_j, \quad \forall t \quad LT_j(t) \leq ELT_j(t)$$

ou encore $LT^{(i)}$ est toujours correct $\forall i : i = 0, 1, 2, \ldots$

11.3 Exactitude totale

Intuitivement, l'exactitude totale d'un algorithme signifie que l'algorithme satisfait sa spécification. Elle se rapporte aux propriétés de *vivacité* ("liveness property"). Formellement, une propriété de vivacité est formulée en termes de but ou d'assertion qui pourrait être vraie au cours de l'exécution de l'algorithme.

Pour notre algorithme, l'assertion se traduit par le fait que l'estimation de la date locale sûre LT croît de manière monotone lorsque sa valeur ELT avance. Ainsi, il nous revient à prouver que: *l'estimation LT_j d'un PL_j par "l'algorithme de LT" croît de manière monotone lorsque sa valeur exacte ELT croît.*

Preuve de la vivacité

En prouvant l'exactitude partielle de l'algorithme dans le cas I, nous avons prouvé, en réalité, que les $LT^{(i)}$ estimés $\forall i : i = 0, 1, 2, \ldots$ croissent de manière monotone. Plus précisément, nous avons prouvé que la condition:

$$LT^{(i+1)} \geq LT^{(i)} \quad \forall i \geq 0$$

est toujours vraie.

Il reste à prouver que *lorsque la valeur exacte ELT d'un PL augmente, l'algorithme le détecte et fait tôt ou tard la mise à jour de la valeur estimée LT du PL. Ensuite, PL propage en aval sa nouvelle valeur estimée.*

La propagation en aval de la nouvelle valeur estimée se fait par l'envoi de la nouvelle valeur LT à l'ensemble de ses successeurs. Il suffit de prouver que, tant qu'il y a une modification de la valeur exacte dans un PL, ce dernier la détecte et fait la mise à jour de sa valeur estimée. Donnons une formulation plus explicite de la proposition à prouver.

Soient P: l'ensemble des PLs dans le système; $ED^k = ED(PL_k)$: l'ensemble de dépendance du PL_k; un $PL_j \in ED^k$ noté par PL_j^k.

Notre raisonnement se rapporte au processus logique PL_j^k. Pour des raisons de clarté, nous abrégeons la notation$LT_j^k(t)$ en $LT_j(t)$. On désigne la $i^{ième}$ estimation de $ELT_i(t)$ à l'instant $t = t_i$ par $LT_j^{(i)}$, alors il revient à prouver que:

si $ELT(t)$ *subit un changement à l'instant t, alors sa valeur estimée LT doit aussi subir un changement à un instant ultérieur t', $(t' \geq t)$.* Ensuite le processus doit propager cette nouvelle valeur estimée à ses successeurs.

Par définition:

$$ELT_k(t) = \min\left\{\min\{lvt_j^k(t), \forall j : PL_j \in ED^k\}, \min\{ts(m), m \in T^k(t)\}\right\}$$

Supposons qu'à l'instant t, le PL_j^k change sa valeur exacte ELT_j et prouvons qu'à l'instant $t' \geq t$, le PL_j^k ressent la modification et change sa valeur estimée $LT_j(t')$.

Désignons par m le message d'événement qui détermine à l'instant t la valeur exacte de $ELT(t)$, on a: $ts(m) = ELT(t)$. Selon la définition de $ELT(t)$ donnée ci-dessus, si $ts(m) = ELT(t)$:

1. soit il existe un PL_j^k dans lequel m est en cours de traitement, ce qui signifie que: $lvt_j(t) = ts(m)$. Mais comme la définition de LT_j prend en compte ce $lvt_j(t)$ modifié, le changement est immédiatement pris en compte dans sa prochaine estimation. c.q.f.d

   Ensuite, PL_j^k réévalue sa nouvelle valeur dans le jeton qu'il expédie à ses successeurs (ou aux PLs en aval de PL_j^k).

2. soit il existe un message m en attente de traitement par le PL_j^k. Ce message m est:

   a. soit dans $UQ_j^k(t)$ ou encore $m \in UQ_j^k(t)$
      dès la prochaine estimation le PL_j^k prend en compte le changement à l'instant t' car la définition de l'estimation LT_j^k fait appel au minimum des messages non-traités de $UQ_j^k(t)$. c.q.f.d.

   b. soit dans un canal (PL_l, PL_j)
      D'autant que le temps de communication est fini, le message parviendra au PL_j^k en temps fini. Désignons par t' l'instant où PL_j^k reçoit le message dans son $UQ_j^k(t)$. Dès cet instant, l'analyse de cas rejoint le cas précédemment étudié.

Résumons

La valeur de ELT croît toujours car sa définition repose sur le minimum entre la plus petite estampille des messages non-traités dans le sous-système ED^k et la plus petite date des dates virtuelles courantes. Ainsi, cette valeur ELT est à l'abri d'un éventuel rollback qui pourrait survenir dans le sous-système ED^k.

De plus, on vient de prouver qu'une fois que la valeur exacte ELT change dans un PL, elle entraîne aussi la modification de sa valeur estimée LT.

11.4 Conclusion

Par la démonstration de l'algorithme LT, nous avons prouvé qu'il est possible de simuler un modèle dans une simulation répartie optimiste sans faire appel à la détermination explicite d'un état global cohérent. Cette nouvelle approche de synchronisation dans un environnement réparti offre de nouvelles perspectives de recherche dans un système réparti. L'algorithme LT nécessite une phase de pré-traitement pour identifier les composantes fortement connexes du réseau simulé.

Il serait intéressant de poursuivre la recherche entreprise afin:

- d'étudier la manière par laquelle on pourrait étendre l'algorithme aux canaux ne satisfaisant pas la propriété PAPS;

- de faire une étude formelle (mathématique) pour déterminer la complexité au mieux et au pire de l'algorithme.

Partie IV

Simulation multi-agents et simulation par événements discrets

Chapitre 12

Un Modèle d'agent pour la SED

Introduction

Le but de ce chapitre est la définition d'un modèle d'agent dédié à la simulation répartie optimiste. Ce modèle d'agent se base sur le concept de la *simulation répartie dynamique optimiste* (développée dans la deuxième partie) pour proposer un modèle d'**agent logique ou AgSED**, capable d'évaluer dynamiquement sa solution courante et de résoudre le problème de conflit de causalité en coopérant avec les autres agents. Un agent AgSED dispose, dans sa structure interne, d'une fonction appelée "qualité de synchronisation" lui permettant de retenir, à chaque estimation de sa *date locale sûre* (présentée dans la troisième partie de la thèse), la meilleure solution. Cette fonction est définie de manière à obtenir, du système simulé, un comportement global cohérent et efficace, en la maximisant au sein de chaque agent.

Le chapitre est structuré en cinq sections. La première présente le principe, de la relaxation restreinte [LE81], utilisé pour définir le modèle d'agent logique AgSED. Un agent AgSED comporte deux parties: l'une traite l'activité individuelle de l'agent AgSED et l'autre traite le fonctionnement de l'agent en tant que composant de la société. La deuxième section et la troisième traitent respectivement de l'activité individuelle et collective des agents AgSED. La quatrième section présente le modèle de résolution de conflit de causalité, inspiré de la simulation dynamique optimiste (développée dans la deuxième partie).

12.1 Relaxation restreinte et simulation optimiste

Le paradigme de la relaxation [LE81] permet à un agent de faire connaître une information en la communiquant à tous ses agents voisins. Si le voisin est intéressé par cette information, il la mémorise puis la transmet à ses propres voisins; il la détruit s'il la juge inutile ou erronée. L'information est ainsi propagée de proche en proche. Ce mode de transmission ne nécessite pas de connaître les compétences des autres agents. Il est meilleur que la diffusion totale car il permet de corriger les erreurs au cours de la transmission de l'information aux voisins proches.

La relaxation restreinte [CG95] est une variante du paradigme de la relaxation où un agent rediffuse ses connaissances à des voisins particuliers de manière à fournir un résultat pertinent à certains de ses voisins sélectionnés.

Un AgSED diffuse ses connaissances à ses proches voisins de manière à satisfaire la requête d'un agent ou à fournir un résultat pertinent à un groupe d'agents donnés. Le concept de la qualité de synchronisation (cf. page 73) est repris dans le cas de la simulation répartie optimiste pour fournir aux voisins proches des informations cohérentes et pertinentes. Les informations sont en SED les messages d'événement et les messages de contrôle. A la réception d'un message par un AgSED, soit il le juge cohérent alors il le traite et transmet plus loin ses répercussions, soit il le juge incohérent, il cherche à rétablir une situation interne et externe cohérente sans erreur de causalité. Le temps nécessaire à communiquer une information à travers le réseau simulé peut être pénalisant, car on accuse un peu de retard pour propager une information dans l'ensemble du réseau.

Un agent AgSED est conçu de manière à ce que:

- la mise à jour dynamique de son état permette d'accroître les performances du simulateur;
- l'agent puisse identifier les conflits de causalité enventuels et de les résoudre en rétablissant une situation cohérente (sans erreur de causalité).

Un AgSED effectue des communications uniquement dans un état cohérent et si son état interne a été modifié depuis le précédent état cohérent. Un AgSED détermine de façon autonome l'instant de la communication car il est seul à connaître l'instant où son un état est cohérent. Cela garantit la cohérence interne de ses données et des messages qu'il élabore pour les autres AgSED. Il en résulte une cohérence collective dans le raisonnement.

12.2 Agent: composant individuel

12.2.1 Représentation d'un agent AgSED

La représentation des autres agents et du modèle simulé est explicitement définie en termes de croyance dans la représentation interne de l'agent. La représentation de soi est également disponible sous forme de la représentation explicite des croyances. En fait, les croyances regroupent les caractéristiques du processus physique simulé par l'agent AgSED.

Représentation de soi-même

Un agent logique AgSED dispose des croyances suivantes:

- *name*: pour identifier l'agent AgSED;
- *inputQueue*: la liste des messages d'entrée;

- *outputQueue*: la liste des messages de sortie;
- *stateQueue*: la liste des états sauvés;
- *SafeQueue*: la liste des opérations définitives sur laquelle on opère les études statistiques et les opérations E/S;
- *LVT*: la date virtuelle locale;
- *LT* sûre ou GVT_{local}: la date locale sûre;
- $Q_{synchro}$: la fonction de qualité de synchronisation. Elle renferme la compétence et le savoir-faire de l'agent (cf. page 73). Elle définit l'activité ou l'interprétation d'un message d'évènement pour en engendrer éventuellement d'autres pour la suite du traitement.
- *currentState*: l'état courant;
- *computeLT*: l'adresse du prochain AgSED en suivant le chemin de recouvrement de la composante fortement connexe. dans ma composante fortement connexe (cf. 10.5.1).
- *goToHead*: l'adresse du prochain AgSED pour se rendre à l'élément de tête (ou l'agent structurel) dans ma composante fortement connexe (cf. 10.5.1).

Bien entendu, pour tout AgSED, tous les champs ci-dessus ne possèdent pas toujours de valeur attribuée. Les champs non-utilisés sont initialisés à une valeur nulle.

Représentation des autres agents

Pour percevoir le monde extérieur, un AgSED (agent logique) dispose d'agent d'interface en dialogue avec l'utilisateur. Et pour percevoir les AgSED agents voisins, il dispose des croyances suivantes:

- *Cin*: l'ensemble des canaux d'entrée (les prédécesseurs) du processus physique associé dans le modèle simulé;
- *Cout*: l'ensemble des canaux de sortie (les successeurs) du processus physique associé dans le modèle simulé;
- *sccIn*: ("strong connexe composant input channels") la liste des canaux en entrée de la composante fortement connexe interne à l'agent. Cette liste sccIn est un sous-ensemble de *Cin* (cf. 10.5.1);
- *sccOut*: ("strong connexe composant output channels") la liste des canaux de sortie de la composante fortement connexe interne à l'agent. Cette liste sccOut est un sous-ensemble de *COut* (cf. 10.5.1);

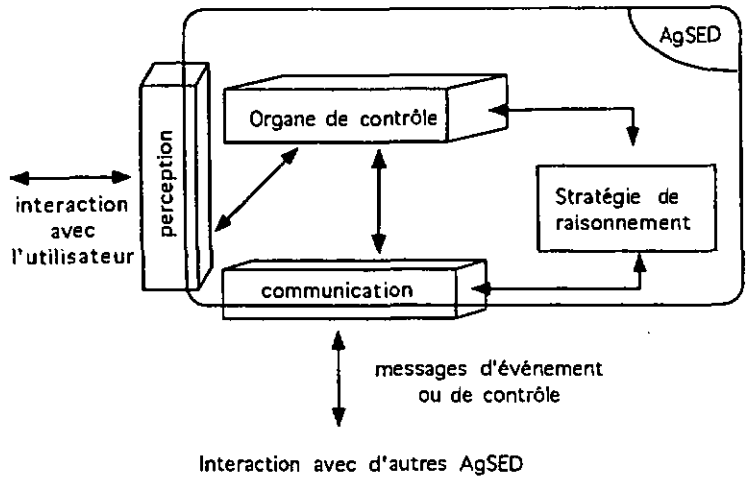


Figure 12.1: Architecture d'un AgSED (agent logique)

Autonomie d'un AgSED

Un agent AgSED possède un ensemble de croyances sur les compétences d'autrui. Il connaît les autres agents AgSED qui peuvent s'intéresser à ses déductions. Il communique des informations nouvelles aux agents qu'il juge concernés sans sollicitation. Ces informations sont transmises par les messages d'événement ou de contrôle. C'est un de ses aspects autonomes. Par exemple, on retrouve ce mode de communication autonome dans le monde naturel. Les fourmis déposent des phéromones sur le chemin qui les ramène d'une source de nourriture vers la fourmilière. D'autres fourmis détectant l'émission, pourront collaborer à l'approvisionnement de la collectivité.

Les décisions prises par un AgSED sont influencées par sa qualité de synchronisation.

12.2.2 Architecture d'un agent AgSED

Un agent AgSED [GD95](cf. figure 12.1) manipule les connaissances qui ont trait à l'existence propre de l'agent dans l'environnement. Il enchaîne les opérations liées au **contrôle individuel** réalisant le contrôle local, et celles liées au contrôle social réalisant **le contrôle global** du système par événements discrets. L'architecture d'un agent est composée de quatre modules:

- le module de communication pour émettre, recevoir et interagir avec les autres agents AgSED;
- le module de perception pour communiquer avec le monde extérieur (l'utilisateur du programme y compris);
- le module de comportement (incluant le raisonnement et le contrôle) pour définir la stratégie d'inférence sur l'état courant menant à une meilleure solution.
- le module de contrôle pour piloter et coordonner les activités des trois modules ci-dessus.

Module de Communication/ Perception

Les communications entre les agents sont réalisées par envoi de messages asynchrones. Dans le système, il existe deux types de communication: la communication point à point (d'un agent à un autre) et la communication par diffusion à un ensemble d'agents AgSED. L'activité de communication est régie par un langage de communication commun à tous les agents AgSED de la simulation et permet l'échange d'informations entre les agents. Chaque agent AgSED dispose d'un organe de traduction (dans son module de communication et de perception) pour identifier le genre d'information: message de contrôle ou message d'événement.

Les messages externes transmis provenant de l'utilisateur constituent les messages perçus par AgSED. Ensuite, ils sont traduits en hypothèses, croyances, buts ou plans de l'agent par son module de perception. Alors que les messages reçus par un AgSED, proviennent uniquement des ses AgSED voisins.

Le module de perception et le module de communication placent les messages d'événements perçus ou reçus dans une liste de messages d'entrée de l'agent. Ces événements externes sont traités par le module de raisonnement (décision et stratégie de raisonnement) qui produit de nouveaux événements. Ces derniers sont placés dans une liste de message d'événements en sortie de AgSED. Le module de communication (resp. de perception) saisit les événements de la liste des événements en sortie pour les envoyer aux autres agents (resp. aux utilisateur) du système.

Module de Contrôle

Le cycle de contrôle est initialisé par un événement extérieur provenant soit du monde extérieur (de l'utilisateur), soit des autres (agents logiques) AgSED du système simulé. Le contrôle se déroule en trois étapes: l'identification des opérations potentielles déclenchées par les messages d'événements; la sélection de la prochaine opération, à partir de la qualité de synchronisation (plan de contrôle); et l'exécution des opérations sélectionnées, qui produit de nouveaux événements.

La fonction "qualité de synchronisation" est un ensemble structuré d'optimisation des paramètres de contrôle de la simulation répartie optimiste (cf. chapitre 7). Chaque paramètre de contrôle décrit une classe d'optimisation que l'agent souhaite accomplir à chaque instant. Cette fonction utilise les indices de performance (paramètres de contrôle) pour évaluer dynamiquement la qualité instantanée de la synchronisation. La fonction fournit donc une représentation explicite du comportement de l'agent logique AgSED.

Module de Raisonnement

Nous considérons la simulation optimiste comme un ensemble d'agents interconnectés selon la structure de communication du modèle simulé. Pour rendre le simulateur optimal, nous cherchons à rendre optimale l'exécution de chaque agent en modifiant "adaptativement" ses paramètres de contrôle. Le raisonnement est associé à une représentation de connaissances, de compétences et de stratégies. Il est exprimé par la fonction "qualité de synchronisation".

Stratégie de raisonnement

Si les messages d'événements reçus ou perçus ne remettent pas en cause le raisonnement en cours, le module de contrôle engendre des messages d'événements chargés de simuler les opérations externes de l'agent. Ces messages d'événements sont placés dans la liste des messages de sortie de l'agent (*OutputQueue*).

Si les messages d'événements reçus ou perçus remettent en cause le raisonnement en cours (messages désynchronisés), l'activation du module de raisonnement définit un nouveau modèle de raisonnement pour rétablir un état cohérent. La synchronisation perdue est rétablie par un AgSED, en invoquant le mécanisme de rollback pour rétablir son état cohérent interne et l'envoi des antimessages à ses voisins pour rétablir l'état cohérent externe.

Un agent AgSED met en œuvre un mécanisme de contrôle dynamique pour structurer ses actions internes ou externes de façon cohérente et pertinente, en s'appuyant sur sa qualité de synchronisation (cf. chapitre 7). Pour expliquer le mécanisme d'inférence du raisonnement, nous allons représenter un agent par une *machine à états finis ou un automate fini* qui dépend du temps.

Un agent AgSED comme un automate fini

Pour modéliser le fonctionnement d'un AgSED en des points discrets de la simulation, on peut décrire son comportement au cours du temps à l'aide d'une machine à états finis. Selon la terminologie de Minsky [Min67], cette machine est représentée par les équations suivantes:

$$Q(t+1) = Q_{synchro}(Q(t), S(t))$$

$$R(t+1) = Q_{synchro}(Q(t), S(t))$$

où $Q(t)$: représente l'état interne courant de la machine à l'instant t; $S(t)$: représente les paramètres d'entrée ou stimuli à l'instant t; et $R(t)$: représente les paramètres de sorties ou réponses à l'instant t.

Un AgSED est donc modélisé grâce à ses paramètres d'entrée, ses états internes, ses paramètres de contrôle et la fonction $Q_{synchro}$. La fonction $Q_{synchro}$ correspond aux mécanismes de "raisonnement" qui lui permettent, selon les stimuli qu'il reçoit (perception/communication), de réagir pour modéliser ses croyances, ses intentions et de produire des réponses. Ces lois propres à l'AgSED définissent son autonomie de décision.

L'état du monde de l'agent est défini, à tout instant, par le triplet suivant: (Pa, Sa, A) où Pa la perception du monde extérieur, (l'entrée de l'environnement vers l'agent et la réception d'un message d'un autre agent); Sa représente le savoir spécifique de l'agent, ses croyances et ses intentions; et A l'action de l'agent sur l'environnement extérieur.

Le système de simulation par événements discrets dans un environnement multi-agent (SEDEMA) est composé d'un nombre fini d'agents. Chaque agent est une machine à états finis. Le système multi-agents par événements discrets (SEDEMA) peut donc être modélisé par des automates finis.

12.3 Modèle social de l'agent AgSED

La possibilité de travailler collectivement, sous-entend deux modes de fonctionnement; l'agent AgSED peut décider:

1. de résoudre son problème localement tout seul;
2. de résoudre son problème avec d'autres agents AgSED en sollicitant l'aide de ses prédécesseurs (voisins).

12.3.1 Coopération

Pour coopérer, chaque agent AgSED est en communication avec ses voisins (prédécesseurs pour recevoir et successeurs pour émettre des messages d'événement). Il dispose de connaissances sur les autres et en particulier, sur la qualité de l'activité de ses prédécesseurs, grâce aux messages d'événement ou de contrôle reçus. L'agent se sert de ces messages pour évaluer la contribution de ses prédécesseurs à sa qualité de synchronisation. L'évaluation de la qualité de synchronisation à chaque cycle de GVT définit dynamiquement la coopération entre les agents. Pour limiter les événements incertains, un agent AgSED dispose d'un mécanisme dynamique de contrôle local (cf. chapitres 7 et 8) basé sur la qualité de synchronisation de l'agent avec ses agents voisins.

Un AgSED ne diffuse de l'information qu'aux agents susceptibles d'être intéressés par celle-ci. Chaque AgSED utilise ce mode de transmission "sélectif " afin d'avoir un comportement visant à aider les agents-sources à l'origine de la relaxation restreinte. Dans la simulation par événements discrets, la détermination des agents intéressés par une information est fournie par le graphe du réseau des PPs. Il représente, dans le modèle multi-agents, le graphe du réseau de croyances et des connaissances. Malgré le parcours sélectif du graphe des connaissances sociales, il se peut que certains agents attendent très longtemps l'information souhaitée (attente du jeton sur les canaux d'entrée). La diffusion des jetons peut être très longue en raison d'une saturation d'une partie du réseau par les messages transmis.

L'état instantané d'un agent lui indique s'il faut attendre la terminaison d'une action ou l'obtention d'information supplémentaire pour accomplir un acte. Nous pensons tout particulièrement à la règle d'attente des jetons sur les canaux d'entrée d'un agent AgSED d'après l'algorithme LT sûr. Un agent AgSED est donc capable de juger l'opportunité de communiquer ses solutions partielles à ses successeurs.

12.3.2 Organisation

Dans un environnement réparti de ressources (processeurs) limitées, le contrôle social est explicite. Il définit le protocole d'interaction et l'organisation entre les agents AgSED. Les protocoles d'interaction définissent la séquence des échanges de messages d'événement entre les agents par rapport à la résolution du problème simulé, alors que les organisations définissent des interactions possibles des agents sélectionnés entre eux en vue d'améliorer la qualité de synchronisation. Cette organisation intervient dans l'auto-ordonnancement

des agents logiques placés sur un même processeur. Par ailleurs, le contrôle des jetons forme aussi une organisation où les agents d'une composante fortement connexe coopèrent pour déterminer leur date locale sûre.

12.3.3 Apprentissage et révision des croyances

L'autonomie des agents induit nécessairement des erreurs de causalité qui doivent être résolues. L'intégration des agents réactifs d'interface avec l'utilisateur impose à chaque agent AgSED de trouver d'une part, les moyens d'acquérir de nouvelles connaissances et croyances, et d'autre part, l'aptitude à réviser ses connaissances par rapport au dernier état cohérent de ses connaissances.

Comme dans une simulation optimiste, les messages d'événement traités peuvent être mis en cause par un rollback; les croyances et leurs résultats établis peuvent être également mis en cause. C'est pourquoi, un agent AgSED dispose d'un mécanisme de révision de croyance destiné à la gestion des conflits de causalité en vue de rétablir un état cohérent, en particulier entre les connaissances reçues et les connaissances expédiées.

12.4 Résolution de conflits de causalité

Dans un système multi-agents par événements discrets (SEDEMA), où chaque agent n'a structurellement qu'une connaissance partielle et incomplète de la société et des compétences d'autrui, le conflit de causalité devient inéluctable. Un modèle dynamique (partie II) de résolution des conflits basé sur le compromis local et les indices de performances a été proposé (cf. chapitre 7). Ce modèle tente de maximiser à chaque estimation de la date locale sûre, la qualité de la solution partielle en maximisant la fonction $Q_{synchro}$ de l'agent.

La résolution d'un conflit de causalité s'effectue en trois étapes:

1. **Détection du conflit**

   Un agent AgSED est en conflit avec autrui s'il reçoit une information (message d'événement désynchronisé) qui met en cause au moins une de ses conclusions. Dès qu'un conflit survient, il procède à la résolution de conflits de causalité internes et externes.

2. **Résolution de conflits internes**

   Elle consiste à rétablir une situation interne sans conflit de causalité. Un AgSED rétablit un état cohérent sans erreur de causalité à l'aide du rollback (chapitre 4). Cela suppose que l'agent AgSED a la capacité d'élaborer un raisonnement pour restaurer des états cohérents qu'il aurait sauvés. Il est capable d'étudier les conséquences des nouveaux états engendrés sur ses connaissances et croyances.

3. **Résolution de conflits externes**

   La résolution externe se rapporte aux répercussions externes sur les agents voisins AgSED de l'agent qui a reçu le message désynchronisé. L'AgSED récepteur du

message de désynchronisation envoie des antimessages à ses voisins qui déclenchent un conflit chez les récepteurs. En fait, la réception d'un antimessage remet en cause au moins une croyance pouvant entraîner un raisonnement similaire de résolution de conflits.

12.5 Conclusion

La relaxation restreinte définit un comportement de contrôle individuel où chaque agent peut décider de la pertinence du résultat mis à disposition à la société d'agents. La relaxation restreinte de la simulation optimiste dynamique est indépendante de la sémantique véhiculée dans les messages, ce qui rend flexible l'approche dans d'autres systèmes multi-agents existant sans entrainer trop de modifications.

Le modèle d'agent AgSED bénéficie de l'avantage du contrôle dynamique local et de celui de la date locale sûre dans la distorsion temporelle. Ces avantages accumulés lui permet de traiter efficacement les messages d'événement et les messages de contrôle, à partir de son état local. Ainsi un agent AgSED est un agent autonome, au sein d'une société capable, de raisonner, de coopérer avec ses voisins, de résoudre les situations conflictuelles avec ses voisins, d'apprendre et de réviser ses croyances, en vue de rendre efficace le travail collectif de sa société.

Chapitre 13

Simulation par événements discrets Multi-agents

Introduction

Face à la complexité des interactions d'un système (phénomène réel), le concepteur de la SED ne dispose pas toujours de la méthode ni de critères pertinents pour évaluer la performance du système. Nous proposons une modélisation multi-agents (MA): un simulateur par événements discrets multi-agents (SEDMA) pour simuler le comportement du système d'étude. Le simulateur prend en entrée le réseau simulé (la description du système physique simulé), sa condition d'arrêt et un jeu de paramètres d'entrée. Le simulateur MA simule le comportement d'ensemble du système pour fournir en retour les mesures pertinentes relatives à sa performance.

Le but de ce chapitre est la définition d'un univers multi-agents dédié à la simulation répartie optimiste en utilisant le modèle d'agent AgSED défini au chapitre précédent. Ce paradigme est basé sur le concept de la répartition de l'intelligence entre les agents, ces derniers formant une société, dans laquelle chacun a une certaine autonomie.

L'étude de la simulation multi-agents par événements discrets comporte six sections. La première passe en revue la motivation qui mène à l'étude de la simulation d'un système. La deuxième présente le simulateur multi-agents par événements discrets. La troisième section présente les critères de réalisation de l'environnement SEDMA. La quatrième presente la réalisation de l'environnement de programmation de SEDMA en Smalltalk. la cinquième la différents modes de terminaison du simulateur réparti.

13.1 Simulation d'un système réel

Dans la pratique, il existe quelquefois des problèmes qui ne peuvent se résoudre par un algorithme. C'est en présence de ces problèmes que la technique de la simulation par événements discrets se propose comme le suppléant à l'expérimentation et à la démonstration pour trouver une solution. Cette section aborde la problématique de la modélisation et de la simulation multi-agents. Ces problèmes (ou systèmes réels) peuvent être décomposés

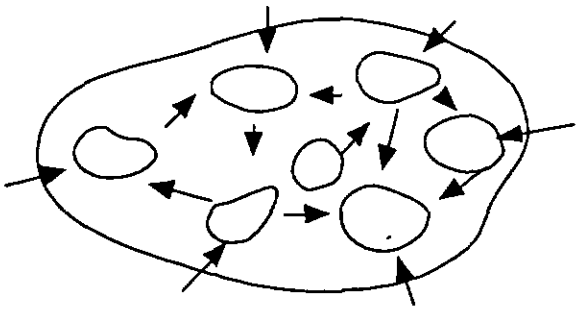


Figure 13.1: Système réel et ses sous-systèmes

en sous-systèmes élémentaires et indépendants ayant des objets (passifs) entrants et sortants (selon la figure 13.1). Les objets passifs, appelés parfois **clients** constituent les seuls moyens dont les sous-systèmes internes disposent pour échanger de l'information. Tous les sous-systèmes fonctionnent en parallèle.

Rappelons que la technique classique de la simulation par événements discrets vise à construire une abstraction de la réalité et à la faire évoluer à tout instant où un changement intervient dans le système étudié. Cette technique offre plusieurs avantages dont les plus marquants s'inscrivent dans les cadres suivants:

- Elle est une excellente méthode de comparaison, d'amélioration et d'appréciation des risques dans le cadre du développement d'un futur projet. Elle donne une connaissance des profits ou des pertes en remplaçant une expérimentation hasardeuse et coûteuse.
- Elle offre un domaine expérimental d'essai pour apprécier de nouvelles idées, ainsi que pour prévoir les conséquences de changements imposés, souhaités ou simplement essayés sur un système, afin de mieux comprendre son fonctionnement et ses principes. Elle permet d'évaluer la performance d'un système non construit avant de passer à la phase de réalisation.

La simulation par événements discrets multi-agents a pour but de décrire aussi finement que possible la structure du problème à étudier, chaque fois qu'un changement survient dans le système simulé. Elle vient en aide pour déterminer:

- le jeu de paramètres d'entrée, adapté au problème;
- les comportements ou modes de fonctionnement adéquats aux entités (agents) pour que le système ait le comportement global cohérent souhaité;
- les lois de répartition correspondant au mieux à la fréquence des arrivées et à la répartition des temps de service aux divers centres de service.

L'étude de la simulation multi-agents par événements discrets n'est pas un processus strictement séquentiel. Cependant, elle comporte trois grandes phases qui ne se recoupent pas dans le temps: la construction du modèle multi-agent, la programmation du modèle

exécutable (simulateur multi-agent) et l'analyse des résultats pour comprendre et prévoir le comportement du phénomène réel. Notre étude se rapporte à la manière de passer du modèle Multi-Agents (ou système physique classique) à la réalisation du simulateur multi-agents.

13.2 Simulateur Multi-Agents pour SED

Le simulateur Multi-Agent pour la imulation répartie peut alors être considérée comme un ensemble d'agents dynamiques communiquant par messages asynchrones. La technique de contrôle adaptatif est appliquée localement à chaque agent pour maximiser l'efficacité du système entier. Un simulateur par événements discrets multi-agents (SEDMA) est un ensemble d'agents qui simule le comportement d'un phénomène réel chaque fois qu'un changement intervient dans le système réel. L'unité conceptuelle de base du simulateur est l'agent. En effet, le simulateur est composé:

- d'une communauté d'agents réactifs prédéfinis:
  - agents de visualisation,
  - agents utilitaires;
- d'un ensemble d'agents logiques dont la topologie est dictée par le réseau du système physique simulé;
- d'un ensemble d'utilitaires invisibles à l'utilisateur tels que les modèles standards de création, de destruction des agents logiques ou structurels et le modèle standard d'interface d'entrée/sortie des agents (interfaces) à la requête de l'utilisateur;
- d'une base de données propre au système pour sa gestion interne.

Après avoir présenté les différents types d'agents dans le simulateur, cette section étudie les agents fonctionnels. Ces derniers sont spécifiés par l'utilisateur.

13.2.1 Différents types d'agents

Le simulateur comporte: des agents logiques, des agents de contrôle, des agents structurels, des agents statistiques, des agents interfaces et des composants passifs. Seuls les agents logiques sont reliés entre eux par des supports physiques de communication.

Les agents logiques ou AgSED constituent le point de départ de la description fonctionnelle du système. Un agent logique représente un centre d'activité (processus physique) du système. Les agents logiques créent, traitent, manipulent et détruisent des unités passives. Les canaux qui constituent le support physique de communication des unités passives (client ou message d'événement) d'un agent logique à un autre.

Les agents structurels définissent le comportement d'un agent logique au sein d'un groupe. Il définit, à l'intérieur d'une société d'agents, le protocole de communication, de négociation et de coopération.

Les agents interfaces servent à visualiser le fonctionnement des agents logiques du simulateur au cours de son évolution.

agent logique

Un agent logique AgSED désigne un centre d'activité du système étudié. Il modélise les activités d'un processus physique, si l'on considère que le modèle MA est composé d'un ensemble de processus physiques. Un agent fonctionnel désigne un agent que l'utilisateur doit définir. Le rôle de l'interface d'entrée est de définir une manière conviviale pour définir les caractéristiques des agents logiques. Un agent logique exerce des traitements sur les unités passives. Ces dernières sont créées, manipulées et détruites par des agents logiques.

Un canal sert de support physique de communication entre les agents logiques. Généralement, on distingue deux types de canaux: les canaux de contrôle de flux et les canaux de messages d'événements. Les canaux de contrôle de flux servent exclusivement à contrôler (détecter et débloquer les situations anormales) le fonctionnement du système. Les canaux de messages d'événements servent à avancer la simulation en traitant les messages d'événements du système étudié. Dans notre système, on utilise le même canal pour transmettre les messages de contrôle et les messages d'événements.

agent de contrôle

Un agent de contrôle est rattaché à chaque agent logique. Il sert à avancer la date locale sûre d'un PL, à détecter et à rétablir les conflits de causalité. Il évite les situations ou les comportements susceptibles de mener à une impasse pour permettre le calcul de la date locale sûre par chaque agent logique. Il gère les fossiles et les opérations d'entrée/sortie pour visualiser des états cohérents. Les agents de contrôle coopèrent selon la topologie des agents logiques.

agent structurel

Un agent structurel définit le regroupement en organisation structurelle d'agents. Il permet aux agents d'un groupe (d'une composante fortement connexe) de traiter un ensemble d'activités en se concertant. Un agent structurel est l'élément de tête d'une composante fortement connexe, décrit dans le chapitre 10. Il intervient dans le calcul de la date locale sûre de chaque agent logique.

L'autre type de regroupement dans le système est l'ensemble des agents logiques sur un même processeur. Les agents logiques forment le groupe des agents éligibles sur un processeur au cours de l'ordonnancement.

agents statistiques

Les agents statistiques ont pour fonction de mesurer l'activité des agents logiques, des agents de contrôle et des agents structurels. Ils recueillent les données brutes qui servent

à évaluer la performance du système. Ils sont rattachés aux agents logiques et structurels pour relever les données statistiques. Ils ne sont pas connus a priori par l'utilisateur, car ils sont automatiquement engendrés dès la création des agents auxquels ils sont associés. A partir des données brutes ou élémentaires, l'utilisateur peut établir des mesures statistiques plus élaborées telles que:

- la moyenne (de temps de service, d'attente);
- le taux (d'occupation, de messages traités).

Un agent statistique greffé sur un agent logique permet d'évaluer:

- le nombre de messages d'événements reçus et envoyés;
- le nombre de messages désynchronisés et de rollback;
- la taille des différentes listes ("InputQueue", "OutputQueue", "saveQueue") pour évaluer la mémoire utilisée;
- le nombre de messages d'événements (respectivement de contrôle) reçus et envoyés;
- le nombre d'unités passives (clients) traitées;

Un agent statistique greffé sur un canal permet d'évaluer:

- le nombre de messages d'événements et/ou de contrôle ayant passé par le canal;
- le taux d'activité (ou d'inactivité) du canal.

agent interface

Un agent interface est un agent réactif d'entrée/sortie situé entre le moteur et l'utilisateur. Il sert à visualiser le contexte d'exécution du simulateur multi-agent. Il décrit l'état de l'agent auquel il est rattaché. On distingue trois types d'agents interface:

- agent éditeur,
- agent afficheur,
- agent inspecteur.

Un agent interface permet l'affichage de l'état des agents logiques, des agents structurels, des canaux, de la date du simulateur. Ils servent à afficher les données statistiques brutes ou travaillées. Il s'agit, en fait, d'un outil d'interface d'entrée-sortie pour visualiser le comportement d'un système réparti.

La réactivité des agents interface permet à l'utilisateur d'accéder à un agent logique au cours de l'exécution en envoyant une requête au simulateur multi-agents. Ce dernier répond à la requête en retournant son dernier état cohérent.

13.2.2 Mesures effectuées sur le simulateur

Les mesures faites par le simulateur permettent de connaître:

- la communication entre les agents logiques;
- le temps de service;
- le taux d'activité;
- la mémoire occupée.

Mesures sur la communication

Les agents du simulateur communiquent par envoi de messages asynchrones. Le simulateur pourrait fournir, par exemple, des mesures relatives à:

- un agent logique: le nombre de messages d'événement envoyés et reçus; le nombre de messages de contrôle envoyés et reçus ainsi que le nombre de messages traités;
- un canal reliant deux agents: le nombre de messages d'événement et de contrôle ayant transité; le taux d'activité ou d'inactivité; et la répartition du temps de trafic des messages.

Mesures du temps de service

Selon le type et les caractéristiques des unités passives dans le simulateur, il faut déterminer par exemple: le temps de traitement de chaque type d'unité (prioritaire et non-prioritaire) traitée, le nombre et le temps de traitement de chaque type représenté dans les messages d'événement.

Mesures des taux

Les taux sont généralement de bons indicateurs de l'évolution du système dans le temps. On peut par exemple déterminer le taux d'occupation de chaque serveur; le taux d'occupation des canaux et le taux de messages traités.

13.2.3 Comment spécifier un agent logique

La spécification d'un agent est fournie par l'utilisateur.

L'utilisateur, après avoir défini le réseau, doit encore spécifier pour chaque agent logique:

- les données d'entrée de l'agent;
- les données de sortie de l'agent;

- la description du fonctionnement (le moteur) de l'agent,

et, si l'on s'occupe de la gestion de mémoire, son allocation de mémoire, et sur quel site, quelle machine, et quel processeur il faut loger l'agent logique.

13.2.4 Actions d'un agent logique

Les actions possibles d'un agent logique consistent à:

- recevoir des messages d'événement ou de contrôle d'autres agents logiques;
- changer son état interne: manipulation de son état local selon la spécification de sa qualité de synchronisation;
- envoyer des messages d'événement ou de contrôle à d'autres agents;
- traiter les requêtes d'interface de l'utilisateur pour les différents agents.

13.3 Critères de la conception de SEDMA

Les caractéristiques considérées dans la conception de SEDMA par ordre d'importance décroissant sont:

- flexibilité et robustesse
- convivialité et facilité d'emploi
- évolutivité

Flexibilité et robustesse

Pour permettre à SEDMA de supporter une large application de problèmes pratiques, il convient d'envisager une plate-forme adaptée aux divers modèles, en particulier à ceux de la simulation par événements discrets. La construction de la plate-forme doit par définition éviter la construction de plusieurs environnements d'expérimentation. SEDMA doit offrir une couche conceptuelle, destinée à un grand public.

SEDMA doit être assez robuste pour résister aux erreurs de l'utilisateur, même dans des conditions anormales.

Convivialité et facilité d'emploi

L'utilisateur de SEDMA n'étant pas nécessairement un informaticien avancé, la mise en œuvre de SEDMA doit être facile à utiliser. Cette qualité nécessite la présentation de SEDMA où l'utilisateur fournira très peu d'effort pour spécifier son modèle. Une interface graphique sera l'idéale.

Les applications multi-agents sont par nature réparties, que ce soit fonctionnellement ou géographiquement. Pour représenter cette répartition, un système multi-agents doit permettre la définition d'entités locales avec un comportement local et un contrôle local. Il doit également permettre la simulation de l'exécution simultanée de plusieurs interactions entre les agents, que ce soient des actions physiques ou des communications. Pour suivre l'exécution simultanée des actions, le système doit comprendre des outils facilitant la visualisation des actions exécutées par chaque agent.

Evolutivité

La construction de SEDMA doit éviter la mise en œuvre répétitive des éléments communs à plusieurs types d'agents et à plusieurs couches. C'est la raison pour laquelle nous avons choisi une conception modulaire basée sur les mécanismes d'héritages et d'objets. SEDMA doit être conçu de manière à intégrer facilement de nouveaux concepts. Il doit admettre la modification de certains aspects de la plate-forme sans impliquer la reconstruction intégrale de la plate-forme. Sa structure doit être évolutive. Une solution pour faciliter la modification partielle du système consiste à séparer conceptuellement les différentes parties où chaque partie correspond à une organisation modulaire.

13.4 Réalisation de SEDMA en Smalltalk

L'approche multi-agents est attrayante et suscite un intérêt croissant dans des domaines d'application très variés. Cet intérêt s'accompagne du besoin d'environnements de développement pour infirmer ou confirmer les concepts théoriques multi-agents. Pour répondre aux divers besoins, deux types d'environnements se sont développés: les environnements spécifiques et les environnements génériques. Le premier type d'environnement consiste à construire un système multi-agents spécifique représentant le mécanisme de coordination d'un modèle d'agent unique (par exemple, CNet [SD80] et de DVMT [DM89]). Le second type vise à construire un environnement pour réaliser et expérimenter plusieurs modèles d'agents avec différents mécanismes de coopération dans divers domaines d'application. Ce dernier type d'environnement est connu sous le nom de plate-forme multi-agents tels que MICE, MACE, MAGES [Bou93]. Notre plate-forme de simulation par événements discrets multi-agents (SEDMA) s'inscrit dans le second type.

SEDMA est un simulateur réparti pour la simulation par événements discrets fonctionnant dans un environnement réparti simulé. Le noyau ou le moteur de simulateur intégré à l'environnement est un simulateur optimiste. La réalisation de la distorsion temporelle utilise la théorie de contrôle adaptatif pour réduire le nombre de rollback entre deux calculs de la date locale sûre en vue d'équilibrer l'avancement des horloges virtuelles locales LVT.

Le simulateur comporte deux parties distinctes (cf. figure 13.2):

- un moteur de simulation par événements discrets réparti optimiste;
- une interface graphique interactive pour spécifier le modèle simulé et suivre visuellement les critéres de performance du système.

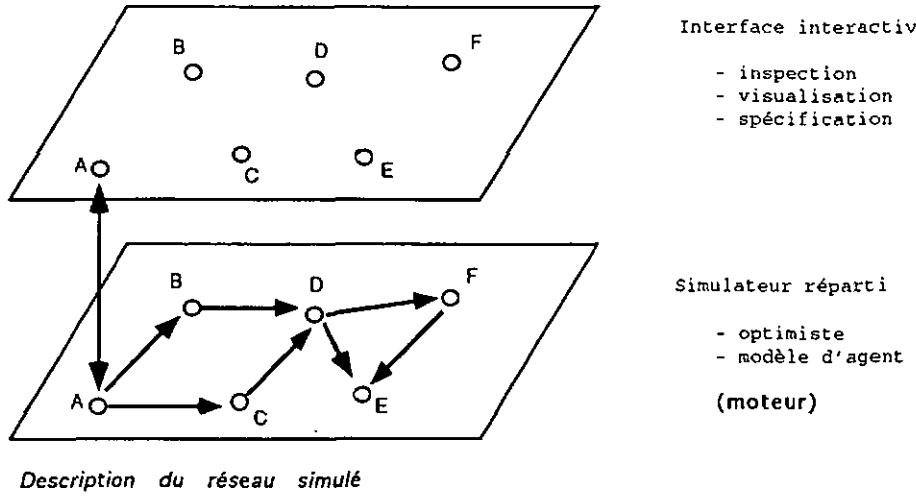


Figure 13.2: Différentes parties du simulateur

Cette section présente les critères de la conception de SEDMA, le moteur ou le noyau du simulateur dynamique optimiste et l'interface de SEDMA conçue pour spécifier interactivement le modèle et suivre visuellement les critères de performance de SEDMA.

13.4.1 Moteur de SEDMA

SEDMA est développé à partir d'un environnement de programmation concurrent basé sur les objets (OBCP), en particulier le système Actalk [Bri94], [Bri89]. La conception de cette plate-forme s'inscrit dans le cadre de plusieurs travaux réalisés en intelligence artificielle distribuée (IAD) autour des modèles objet et acteur [Hew77]. Rappelons le concept de la programmation concurrente basée, sur les objets (OBCP), avant de présenter les couches supérieures qui entrent dans la construction de SEDMA.

Programmation concurrente basée sur des objets

Dans le modèle original d'acteur, toutes les entités manipulées sont des acteurs. Un acteur est une entité active dont l'exécution est concurrente. Les modifications de son comportement se produisent par échange de messages entre les acteurs. La transmission de messages est asynchrone. Elle est réalisée à travers un système de boîte aux lettres. Un acteur communique uniquement avec ses accointances (les autres acteurs qu'il connaît).

Dans le modèle objet (par exemple Smalltalk-80 [AR83], ObjVlisp [Coi87]), toutes les entités manipulées sont des objets. Les objets sont des entités passives dont l'exécution est séquentielle. Les modifications se produisent par échange de messages. La transmission de message est asynchrone. La structure des objets est déterminée par un ensemble de champs, ou de variables. Le modèle objet inclut généralement la définition de mécanismes de structuration basés sur les concepts de classe, d'instance, d'héritage et de variable.

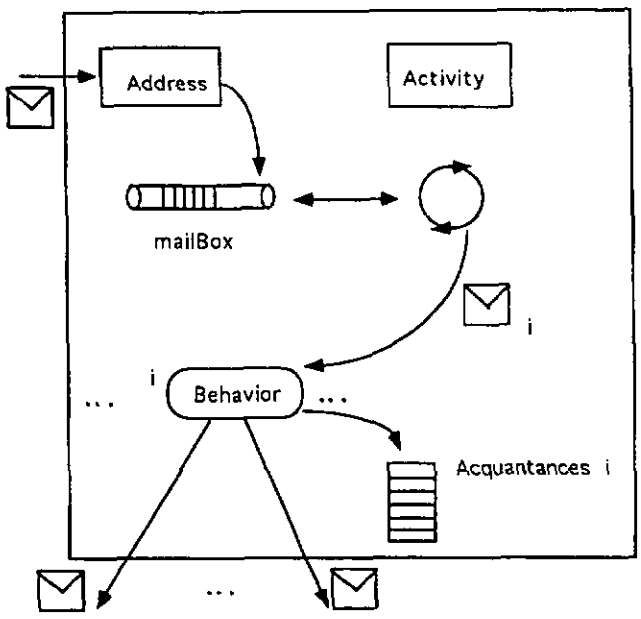


Figure 13.3: Composants d'un acteur **Actalk**

Noyau Actalk

Le noyau de Actalk [Bri94] (acteur dans Smalltalk) étend les objets passifs et séquentiels définis dans Smalltalk-80 en des objets actifs et concurrents, communiquant par le passage de messages asynchrones. Un modèle de communication asynchrone est défini et les acteurs Actalk travaillent comme des objets "sérialisés" [Agh86]. Le modèle de communication synchrone défini dans Smalltalk reste toutefois opérationnel. Ainsi, tous les traitements ne s'exécutent pas nécessairement de façon concurrente. Le noyau Actalk est constitué de trois classes de base qui définissent les mécanismes permettant l'exécution concurrente des objets. La conception d'un nouveau modèle d'acteur ou d'un modèle d'agent est accomplie par la définition de sous-classes à partir de ces classes de base.

Un acteur se compose de trois objets (cf. figure 13.3):

- une *adresse* (une instance de la classe Address) elle reçoit les messages envoyés à l'acteur et les dépose dans la boîte aux lettres;
- une *activité* (une instance de la classe Activity), il s'agit du processus traitant successivement les messages contenus dans la boîte aux lettres;
- un *comportement* (une instance de la classe ActiveObject), il représente l'organe de traitement des messages. Le traitement d'un message dépend de la nature du comportement courant des accointances. Les actions possibles d'un acteur sont: lecture/écriture sur les accointances, la création d'acteur et le remplacement de comportement.

Les trois classes de base (cf. figure 13.3) du noyau de Actalk [Bri94] sont:

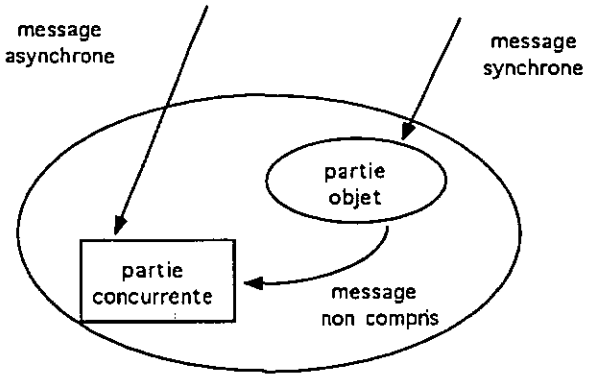


Figure 13.4: Interprétation d'un message actalk

- La classe **ActiveObject** représente le *comportement interne* d'un objet actif, c'est-à-dire celui qui traite les messages; il donne ainsi sa sémantique à l'objet actif. Le programmeur définira dans ses classes de comportements d'objets actifs des programmes concurrents comme sous-classes de ActiveObject et non pas comme classe *Object* (Smalltalk standard). Les constructions spécifiques éventuelles d'un environnement de programmation donné seront également exprimées par les sous-classes abstraites de ActiveObject.

- La classe **Activity** représente l'activité interne d'un objet actif. Cette classe apporte l'autonomie (via l'allocation d'un processus) à chaque objet actif. Elle décrit également la manière dont les messages sont sélectionnés, séquencés et activés. Ceci inclut donc en particulier différentes stratégies possibles de synchronisation de l'activation (également appelé acceptation) des messages.

- La classe **Address** représente l'*adresse* d'un objet actif, c'est-à-dire l'identificateur et la boîte aux lettres associée où vont être envoyés les messages. Elle définit la manière dont la réception des messages sera interprétée, c'est-à-dire différents types possibles de communication.

Pour définir un *objet actif*, le programmeur crée un comportement comme une sous-classe de la classe ActiveObject, puis lui envoie le message actif pour l'activer. Ceci entraîne la création de deux autres composants, l'activité et l'adresse, et le démarrage du processus associé à l'activité.

Interprétation d'un message

Dans Actalk, un acteur est composé de deux parties (cf. figure 13.4), une partie objet à exécution normale dans Smalltalk et une partie à exécution concurrente. Cette dernière comporte les caractéristiques de l'acteur. La transmission asynchrone est réalisée par une redéfinition de la méthode *doesNotUnderstand* qui mémorise les messages non-compris par la couche standard de Smalltalk.

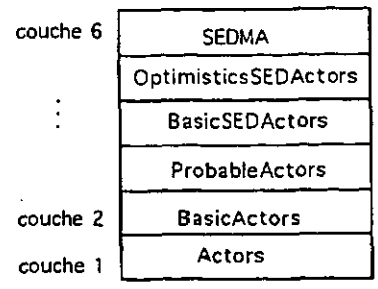


Figure 13.5: Différentes couches de SEDMA

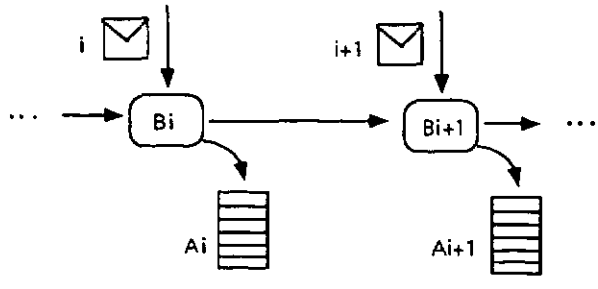


Figure 13.6: Comportement de remplacement

13.4.2 Différentes couches de SEDMA

SEDMA est développé sur un environnement objet, Smalltalk (Objectworks version 4.1). La réalisation de SEDMA comporte quatre-vingts classes Smalltalk répartie conceptuellement en six couches:

La construction modulaire de SEDMA comporte six couches (cf. figure 13.5).

La structure des agents (acteurs) est déterminée par un ensemble de champs. Ces champs ont pour valeur des objets qui représentent les composants de l'agent SED (par exemple l'estampille du message, les canaux de sortie) ou qui déterminent un état de contrôle de l'agent. Les champs d'un agent d'une couche se définissent par la particularisation en ajoutant des champs particuliers d'une couche à une couche supérieure.

La couche 1 constituant la base est déjà étudiée ci-dessus. Etudions l'évolution conceptuelle des autres couches qui mènent à SEDMA.

Acteurs de base: acteur réactif (couche 2)

Cette catégorie de classe comporte essentiellement la particularisation des trois classes de base de manière à rendre réactif (disponible) les acteurs du système. La qualité de la réactivité de l'acteur dépend du remplacement de comportement (cf. figure 13.6). Elle les modifie de manière à étendre le comportement de chaque acteur en vue d'admettre les comportements réactifs. C'est la couche de base pour réaliser le modèle d'acteur (d'agent) de visualisation interactive de la simulation.

En résumé, cette couche particularise les acteurs de base:

- en les rendant *réactifs* (plus disponibles au remplacement de comportement) et en

leurs associant une base d'accointances;

- en définissant les *accointances de base* de l'acteur; cette classe sera particularisée dans les couches supérieures en ajoutant des champs (instances de variable); cette classe comporte la représentation de l'acteur et des autres acteurs avec lesquels ils communiquent;
- en particularisant le *message* de base Smalltalk pour qu'il soit constitué de l'expéditeur, du receveur et du contenu du message.

Cette couche regroupe les classes réparties dans les catégories de classes: SEDMin-Evolution-0, SEDMinEvolution-0-support, SEDMin-Evolution-1.

Les agents logiques sont réalisés à l'aide des acteurs de base (SED).

Acteurs-sondes (couche 3)

Cette couche définit un acteur de base sur lequel peuvent être greffés plusieurs autres acteurs-sondes. Un acteur-sonde est un acteur réactif qui rend service à un acteur de base. Dans certains cas, il établit le lien de communication entre un acteur de base et l'utilisateur. Un acteur-sonde peut être un acteur de visualisation, un acteur statistique.

Cette couche regroupe les classes de la catégorie de classes: SEDMin-Evolution-1.

Les agents de contrôle, agents structurels, les agents statistiques et les agents interface sont réalisés à l'aide des acteurs-sondes.

Acteurs SED de Base (couche 4)

Cette couche de classes particularise un acteur de base pour définir un acteur SED (agent) pour la simulation par événements discrets.

- On ajoute les champs suivants à la *base des accointances* d'un acteur SED:

- l'ensemble des canaux de sortie,
- l'horloge virtuelle locale;

- Chaque acteur est doté d'un comportement qui signale la *présence d'une erreur de causalité* mais ne la traite pas.

- Les messages sont mieux particularisés. Un *message SED* est alors constitué de l'expéditeur, du destinataire, du contenu et de l'estampille.

Cette couche regroupe les classes réparties dans les catégories de classes:

SEDMinEvolution-2-support et SEDMin-Evolution-2.

Acteurs SED optimiste (couche 5)

Cette couche définit les caractéristiques d'un acteur SED pour la simulation par événements discrets répartie optimiste.

- Le comportement d'un acteur optimiste *traite l'erreur de causalité*, la *sauvegarde des états* et la gestion de mémoire des états sauvegardés.

- Un *message* optimiste comporte, outre les caractéristiques des couches précédentes, un *signe* qui peut prendre une valeur négative ou positive.

Cette couche regroupe les classes réparties dans les catégories de classes: SEDMin-Evolution-3, SEDMinEvolution-3-support.

SEDMA ou Agents optimistes (couche 6)

Outre les caractéristiques héritées des couches précédentes, cette couche définit les caractéristiques du modèle d'agent SED. Elle inclut dans son comportement:

- le contrôle adaptatif pour réaliser les algorithmes dynamiques de manière à accroître la qualité de synchronisation des agents SED.
- l'algorithme de la date locale sûre répartie.

Cette couche regroupe les classes réparties dans les catégories de classes: SEDMA-Thesis-Support-Evolution-0, SEDMA-Thesis-Evolution-1 , SEDMA-Thesis-Evolution-2, SEDMA-Thesis-Support-Evolution-1, SEDMA-Thesis-Support-Evolution-2, SEDMA-Thesis-Support-Evolution-3.

Les différentes couches de SEDMIN comporte chacune les classes de tests simples de carwash [Mis86]:

SEDMIN-Tests

SEDMIN-CarWashTest.

Pour vérifier le bon fonctionnement des méthodes et des classes et celles de la couche SEDMA nous avons:

SEDMA-Thesis-Tests

SEDMIN-Thesis-CarWashTest

13.4.3 Collection des données statistiques

Outres les utilitaires ci-dessus, le simulateur met à disposition des classes pour collectionner et analyser des données statistiques engendrées au cours de la simulation. Le simulateur, dans l'état actuel, ne tient pas compte de l'étude de données statistiques en phase d'équilibre [ED96].

Cette catégorie de classes comporte, pour chaque composant (groupe de composants) du réseau simulé, des attributs qui permettent de collectionner et de traiter les informations statistiques. Au cours de l'exécution, l'utilisateur peut, à tout moment, suspendre l'exécution, consulter les résultats partiels, les modifier et relancer l'exécution ou encore terminer l'expérience courante de la simulation quand l'on s'écarte trop des résultats escomptés. Ainsi, les résultats partiels résumant l'objectif de la simulation peuvent être

contrôlés au cours du l'exécution. Cette approche permet de mieux comprendre l'évolution du système simulé et de détecter les conditions sous lesquelles une exécution peut être interrompue.

Remarque 15
Une variante consiste à afficher les données statistiques collectionnés de chaque composant du modèle simulé après n exécutions ou après K cycles de calcul de la date locale sûre par l'algorithme LT.

En résumé, cette catégorie de classes permet à l'utilisateur:

- d'assembler les divers types d'information sur le modèle simulé: les valeurs numériques, les états et les dates associés aux données statistiques;
- de spécifier quand les données accumulées sont cohérentes, quand elles changent, quand un message spécifique est reçu et quand certains messages d'événement sont planifiés;
- de filtrer les données statistiques en affichant la moyenne, les variances des données collectionnées;
- d'afficher les données statistiques sous diverses formes (histogramme, tableau);
- d'écrire les données dans un fichier (ou dans une fenêtre) pour les imprimer en vue de les analyser.

13.5 Terminaison de la simulation

La fin d'une simulation dépend de l'application sous-jacente. Parmi les critères d'arrêt d'une simulation, nous retenons parmi les plus usuels:

- la date finale de simulation est atteinte par l'ensemble du modèle,
- la précision statistique voulue (d'un ou) des paramètres spécifiés à des sous-modèles donnés est obtenue,
- un nombre d'événements spécifié a été simulé à un endroit donné du modèle,
- l'attente d'un événement particulier à un endroit donné du modèle a été simulé.

La condition d'arrêt de la simulation peut être satisfaite par un processus logique (source/sortie) connu à l'avance ou par l'ensemble des agents AgSED (PLs) à une date donnée de la simulation T_{fin}.

Dans le protocole synchrone, l'arrêt des processus logiques ne pose aucun problème car les PLs avancent tous à la même vitesse. Par contre, dans le protocole asynchrone optimiste, les différentes techniques utilisées pour détecter la terminaison se basent sur la date globale virtuelle ou GVT. Dans notre approche, la technique repose sur la date locale

sûre développée dans la troisième partie. L'arrêt des agents logiques AgSED (PL) se base sur la notion de dépendance d'un agent AgSED (PL) vis-à-vis de ses prédécesseurs. Si tous les prédécesseurs d'un AgSED (PL) ont terminé, le AgSED (PL) a aussi terminé et il s'arrête uniquement après avoir reçu de chacun de ses prédécesseurs et après avoir envoyé à chacun de ses successeurs un message de terminaison portant la condition d'arrêt. C'est ainsi que l'ensemble des AgSED (PLs) du système simulé s'arrêtent à partir des sources.

13.6 Conclusion

Le travail s'est concentré sur la modélisation des problèmes de la simulation répartie sur une plateforme multi-agents. La résolution du problème se fait par interaction d'une part entre des agents simulés constituant les processus (centre d'activité) du système physique et d'autre part entre les agents de contrôle de la simulation. Les agents de contrôle, rattachés aux agents logiques, coordonnent leurs activités avec les autres agents pour déterminer la date locale sûre de chaque agent logique. Le respect de la contrainte de causalité est assuré en utilisant le mécanisme de la distorsion temporelle dynamique développé dans la deuxième partie (cf. chapitre 7).

La plate-forme multi-agents par événements discrets offre plusieurs avantages:

- elle accélère le déroulement de la simulation;
- chaque agent logique AgSED influe localement sur le système simulé, comme dans la réalité; le comportement du système simulé est le fruit des interactions entre les agents;
- elle offre la facilité d'intégrer des possibilités d'expression entre les agents: la coopération et la négociation sont plus élaborés que dans la simulation répartie par événements discrets;
- elle offre une nouvelle perspective de recherche de modélisation des systèmes par événements discrets.

L'avantage de la simulation multi-agents pour SED est que, comme dans la réalité, les éléments du système sont évalués aussi bien de manière qualitative que quantitative. Le simulateur multi-agents fournit de l'information pour analyser le comportement qualitatif des PLs, des groupes de PLs et du simulateur entier, par opposition au système classique de simulation qui ne fournit que des valeurs quantitative (statistiques).

Chapitre 14

Etude de Performance de SEDMA sur les Réseaux de files d'attente

Introduction

Un ensemble de personnes faisant la queue à un guichet de ventes de billets est un exemple de file d'attente. Le guichet est alors le serveur de la file d'attente; il "traite" les clients un à un. D'une manière générale, le formalisme des réseaux de files d'attente [Kle75] a été utilisé depuis longtemps pour modéliser les systèmes à événements discrets (par exemple les systèmes informatiques et télématiques, les systèmes de production, le trafic routier). Le système est décrit comme un ensemble de stations-services s'échangeant des clients. Une station-service est constituée d'un ou de plusieurs serveurs. Un serveur représente un processus physique ou centre d'activité pouvant fournir des services équivalents, tandis qu'un client représente un objet passif circulant entre les stations-services.

Un réseau peut être fermé (un nombre fini de clients circulant indéfiniment dans le réseau) ou ouvert (arrivées et départs des clients de ou vers l'extérieur). Les clients peuvent être partitionnés en classes; chacune d'entre elles est particularisée par des caractéristiques de temps de service aux stations et de cheminement entre les files d'attente du réseau. Il est possible, pour une classe donnée, de limiter le nombre de clients circulant dans le système.

La section 1 présente la simulation des nœuds qui interviennent dans la simulation d'un système de file d'attente: les **sources**, les **sorties** (puits du système) et les **nœuds de branchement**. La section 2 présente la structure générale de la simulation à partir du réseau physique. La section 3 contient la présentation de l'analyse quantitative des réseaux de files d'attente. La section 4 définit la spécification de chaque type de nœud et leurs critères de performance. La section 5 présente l'étude de performance de modèles empruntés de la vie pratique et de modèles théoriques complexes pour évaluer la performance du simulateur SEDMA.

14.1 Simulation d'un système de file d'attente

Un système de file d'attente est caractérisé par:

- le processus d'arrivée des clients au système; il décrit la manière dont les clients se présentent dans le système. Le processus d'arrivée est simulé à l'aide des nœuds sources;
- le mécanisme de service des clients aux stations; il est décrit en spécifiant le nombre de serveurs disponibles et par la loi de répartition des temps de service de chaque serveur de la station de service;
- le cheminement des clients d'une station à l'autre; il est décrit en spécifiant la fonction de routage des clients vers les successeurs. On spécifie la prochaine station à laquelle il faut envoyer un client traité;
- la discipline de service des clients à chaque station; il décrit la stratégie pour choisir le client à traiter d'une file d'attente.

On peut aussi définir les limitations de la longueur des files, ce qui entraîne des pertes de clients arrivant à une station-service où tous les serveurs sont occupés ou le blocage d'une file ne pouvant débiter des clients vers une file saturée.

14.1.1 Cheminement des clients

Une activité d'un client se définit par le processus de cheminement d'un client (composant passif) demandant, tout au long de son cycle de vie, l'accès à une succession de stations-service indispensables à sa réalisation. Une activité est réalisée par le cheminement d'un client d'une station à une autre; ce client demande un certain service à chaque serveur. Le cheminement (cf. figure 14.1) d'un client peut être:

- statique: le choix de la station suivante est figé ou toujours le même;
- dynamique: le choix de la station suivante est défini de manière probabiliste ou conditionnelle.

Un client naît à la source et, durant sa vie, chaque client porte une liste de caractéristiques appelée ***signature du client***, regroupant les ordres reçus par le client ainsi que l'identificateur des stations qui lui sont attribuées pour son activité. A chaque passage du client dans une station, l'ordre attribué par cette station est créé ou mis à jour dans sa signature.

14.1.2 Nœud source de simulation

Les sources de clients représentent les entrées du système par lesquelles arrivent les événements extérieurs agissant sur les clients. La source joue le rôle de générateur de clients

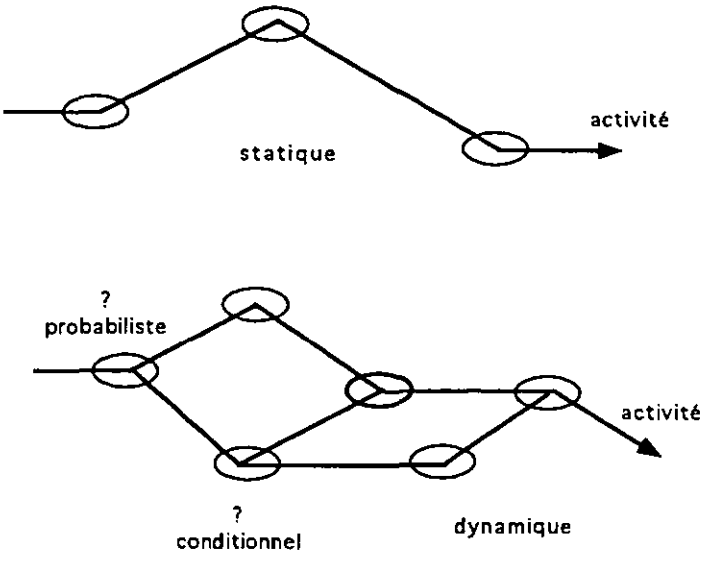


Figure 14.1: Cheminement statique et dynamique des clients

dans le modèle simulé et n'en reçoit aucun. Elle simule la manière dont les clients se présentent au système, plus particulièrement la répartition des intervalles de temps qui séparent l'arrivée de deux clients consécutifs: c'est l'*interarrivée* ("interarrival time"). L'interarrivée peut être définie algorithmiquement; mais, la plupart du temps, elle suit une loi de répartition qui approxime le mieux l'intervalle entre les dates portées par deux clients successifs engendrés par la source. La somme partielle (le cumul) des interarrivées représente la date locale de la source. Une fonction de routage est définie à la sortie de la source pour l'envoi des clients à ses successeurs. Chaque successeur est doté d'une probabilité de recevoir un client.

En l'absence de prédécesseur, le processus simulant la source n'est jamais bloqué par un protocole de synchronisation. Il est nécessaire d'insérer une temporisation réglable dans la boucle principale de la source pour régler la fréquence réelle d'envoi de clients. Une politique d'envoi de messages de contrôle est à définir (par exemple tous les N clients envoyés) pour avancer régulièrement la date locale sûre des différents successeurs.

14.1.3 Nœud puits de simulation

Les puits représentent les sorties du système. Ce sont des nœuds qui ne correspondent généralement à aucune partie du système modélisé mais qui sont utilisés dans la simulation pour éliminer les messages (la sortie d'un client du réseau) et éventuellement pour faire des statistiques sur des clients (quantité, fréquence de sortie etc.). Un client qui sort du réseau est simplement éliminé par un puits du réseau simulé.

14.1.4 Nœud branchement de simulation

Un nœud branchement possède au moins deux canaux de sortie et au moins un canal d'entrée. Il possède également des "poids" ou la probabilité d'emprunter le bon canal de

sortie pour la suite du traitement.

14.1.5 Nœud station-service

Définition des paramètres

Une file d'attente que nous appelons aussi station, notée

$A/S/C/K/L/DS$

par Kendall, est décrite par l'intermédiaire des six paramètres:

- A: processus d'entrée.

  C'est la loi de répartition des intervalles entre les arrivées des clients à l'entrée de la file d'attente.

- S: processus de service.

  C'est la loi de répartition qui régit le service fourni par le serveur de la file.

- C: nombre de serveurs associés à la station.

- K: capacité maximum de la file.

  C'est le nombre maximum de clients pouvant passer dans la file.

- L: population des usagers.

  C'est le nombre maximum de clients pouvant passer dans la station.

- DS: discipline de service.

  C'est la stratégie de gestion des clients de la file d'attente.

Remarque 16

- Les symboles suivants sont utilisés pour définir les principaux processus d'entrée ou de service:

  G: loi générale,

  GI: loi générale indépendante,

  M: loi exponentielle,

  D: loi constante,

  E_k: loi Erlang-k,

  H_k: loi hyperexponentielle d'ordre k,

- Lorsque les trois derniers paramètres de la notation ne sont pas spécifiés, ils correspondent par défaut aux valeurs suivantes: DS = PAPS, $L = +\infty$ et $K = +\infty$.

Politiques de gestion de la file

Les principales disciplines de service dans une file d'attente sont:

- *PAPS: premier arrivé, premier sorti*, (FIFO: first in, first out). Le premier client arrivé dans la file est le premier servi. Un client occupant un serveur le garde jusqu'à la fin de son service.
- *DAPS: dernier arrivé, premier sorti* (LIFO: last in, first out). Le dernier client arrivé dans la file est servi immédiatement. Dans le cas où aucun serveur n'est disponible, il y a préemption. Le client, privé de serveur, reprendra son service au point de rupture (resume) ou dès le début (restart) et seulement après la fin des services de ceux arrivés après lui.
- *Constant (quantum).* Chaque client présent dans la file est servi à tour de rôle par unité de temps fixe (quantum), l'ordre de services étant de type premier, arrivé premier servi.
- *PS: processeur partagé (Processor Sharing).* Tous les clients arrivés se partagent le serveur et sont donc servis en même temps.
- *A: aléatoire (FIRO: first in, random out).*
- *Priorités (avec ou sans préemption).* Les clients sont servis par ordre de priorité. Il y a préemption si un client plus prioritaire arrive après un autre moins prioritaire déjà en service. La reprise de service peut être du type "resume" ou du type "restart" comme dans le cas du DAPS. Dans l'état actuel du simulateur, c'est le générateur qui fixe la priorité d'un client dés sa création et cette priorité ne change pas jusqu'à son élimination. Les priorités peuvent aussi être engendrées dynamiquement.

Lois de service

La durée de service d'un client est une variable aléatoire X qui suit une des lois ci-après. Sa valeur est calculée à partir d'un nombre aléatoire uniforme indépendant U entre 0 et 1. Cette partie est détaillée dans le chapitre 8 [ED96].

Loi constante: $X = CST(Xo)$

X a une valeur constante Xo identique pour tous les clients. X = Xo.

Loi uniforme: $X = UNIF(a, b)$

X a une valeur uniforme entre les deux paramètres a et b.

$X = a + U(b - a)$.

Loi exponentielle: $X = EXP(m)$

Cette loi a une fonction de: $F(x) = 1 - \exp(-\lambda x)$ où λ est un paramètre et $m = 1/\lambda$ la moyenne. Par inversion, on obtient: $X = -\ln(1 - U)/\lambda$; mais comme (U-1) et U ont la même loi, $X = -m\ln(U)$.

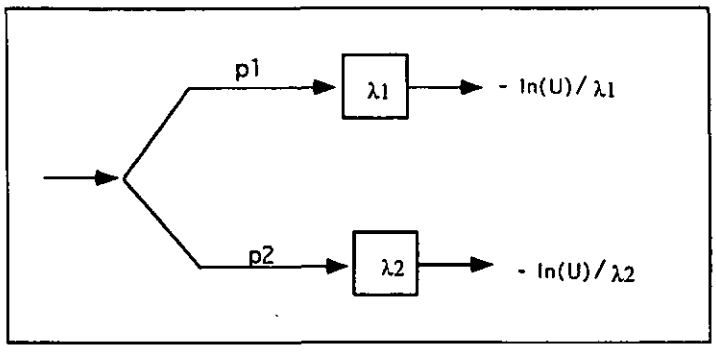


Figure 14.2: La loi hyper-exponentielle

Loi hyper-exponentielle: $X = HEXP(m,a)$

Une variable aléatoire X de la loi hyper-exponentielle est choisie parmi k variables aléatoires indépendantes de la loi exponentielle: une probabilité p_i de choix, i variant de 1 à k, étant associée à chacun d'eux; m est la moyenne et a le carré du coefficient de variation. Ici, nous nous limitons à une loi hyper-exponentielle à deux étages qui permet d'avoir un large coefficient de variation (cf. figure 14.2) et à des valeurs entières de a strictement supérieures à 1.

Loi d'Erlang: $X = ERLANG(m,k)$

Une variable aléatoire qui suit une loi d'Erlang de moyenne m et de k étages est la somme de k variables aléatoires indépendantes de même répartition exponentielle, et chacune de paramètre de λ. La valeur de X déduite de la loi exponentielle décrite plus haut est donc: $X = -\sum_{i=1}^{k} \ln(U_i)/\lambda$. Cette expression peut être simplifiée et donne:

$$X = -(m/k)\ln(\prod_{i=1}^{k} U_i), \quad m = k/\lambda$$

Il faut donc engendrer i variables aléatoires uniformes indépendantes pour le calcul de X.

Fonctionnement d'un serveur

Une station-service effectue elle-même la gestion de ses files d'attentes . Lorsqu'un serveur reçoit un message de la forme:

$< M = (exp, dst, sign, content) >$,

il le dépose dans la file d'attente *inputQueue*. Ces événements sont considérés comme externes. Le traitement d'un événement externe produit des événements internes qui représentent les opérations internes de la station-service. Les événements internes constituent le début de service (ds) et la fin de service (fs) qui indiquent respectivement le début et la fin de service d'un client.

Si deux événements se produisent à la même date, ils sont ordonnés selon leurs priorités pour être traités. Si un serveur est disponible, il débute immédiatement le traitement de son service c'est à dire qu'un ds de même date est engendré pour lui.

La simulation de la fin du traitement d'un événement consiste à envoyer le client traité vers les successeurs et à prendre le premier client de la file pour le mettre en service.

Remarque 17
Une station-service peut comporter des mécanismes de synchronisation (sémaphores, ressources) pour synchroniser l'activité des serveurs entre eux. Dans ce cas, le serveur d'une station-service synchronise avec d'autres serveurs ou avec lui-même (cas de serveurs multiples).

Exemple 5
La figure 14.3 montre un exemple de modèle informatique où deux unités centrales cpu1 et cpu2 se synchronisent à l'aide d'un sémaphore pour l'accès à la ressource disque.

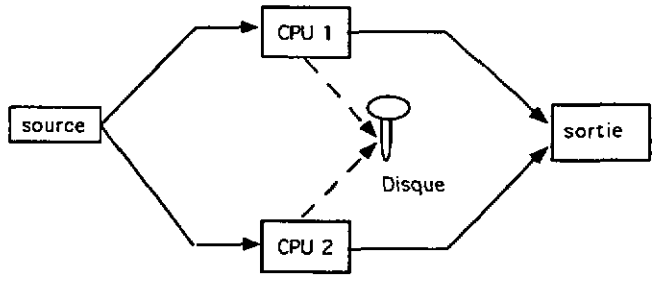


Figure 14.3: un réseau de files d'attente, modèle d'un système

14.2 Structure de la simulation d'un nœud

L'approche de décomposition du modèle de réseaux de files d'attente à simuler se fait selon le nombre de processeurs à disposition. On décompose le modèle en un ensemble de sous-modèles qui seront simulés chacun par un processus séquentiel à placer sur un processeur. Il y a deux approches

1. un PL (processeur) simule plusieurs nœuds du modèle à la fois en utilisant un seul échéancier interne. Cette approche peut s'avérer inefficace si:

   - les nœuds du sous-modèle sont assez indépendants, c'est à dire qu'il y a peu de communication entre eux;
   - les nœuds du sous-modèle ont des liens de communication directs avec l'extérieur.

   En effet, pour faire avancer la date locale sûre d'un PL, ce dernier (sur un processeur) peut être bloqué en attente d'un message de contrôle d'un nœud, alors que d'autres nœuds ne peuvent avancer leur date locale sûre. Par contre, si le sous-modèle est fortement connexe avec peu de liens avec l'extérieur, cette approche est très efficace.

2. La seconde approche consiste à affecter un nœud à un PL de la simulation. Selon la configuration de la machine (ou de l'environnement cible), chaque processeur

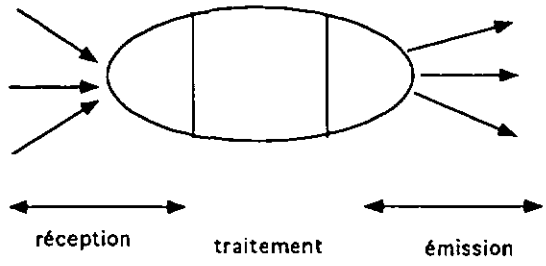


Figure 14.4: Simulateur élémentaire: réception-émission intégrée

peut exécuter un ou plusieurs PL (processus). Chaque nœud simulé avance sa date locale sûre indépendamment des autres et du protocole de synchronisation sous-jacent (risqué ou sûr). Cette approche s'adapte mieux aux modèles génériques de simulation. De plus, sa simplicité de décomposition facilite sa mise en œuvre.

14.2.1 Structure élémentaire d'un nœud

On distingue deux manières de structurer la simulation d'un nœud que nous désignons par: simulateur élémentaire à réception-émission intégrée (cf. figure 14.4) ou à réception-émission séparée (cf. figure 14.5).

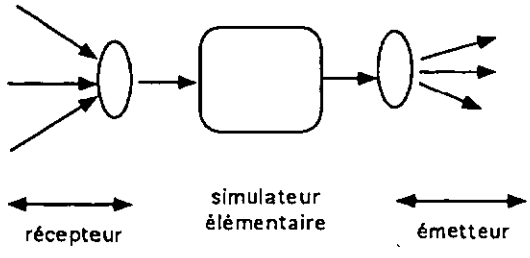


Figure 14.5: Simulateur élémentaire: réception-émission séparée

Si le système des routages des messages est de bas niveau et si le passage d'un message nécessite un rendez-vous entre le routeur local du processeur et les PLs bloquant (c'est-à-dire qu'un envoi bloque forcément l'un des deux PLs expéditeur/receveur), il est préférable de décomposer un nœud à simuler en trois parties physiques qui s'exécuteront sur le même processeur : le *récepteur*, l'*émetteur* et le *simulateur élémentaire*. Ceci permet la mémorisation des messages dans le récepteur et à l'émetteur d'adapter le blocage du nœud de routage et du processus de simulation en fonction du passage des messages. Ainsi, l'organe de traitement du simulateur élémentaire peut être déchargé des *sélections* des messages reçus pour le récepteur ainsi que du filtre des messages expédiés par l'émetteur. Il s'occupe exclusivement du traitement de la simulation de son nœud selon le critère de sélection du récepteur.

Si le système de routage de message entre les processeurs est d'un niveau assez haut pour permettre la mémorisation des messages reçus et émis sans se bloquer et sans bloquer le PL concerné, les différentes étapes (sélection, traitement, entrées-sorties) peuvent être exécutées par un seul simulateur élémentaire.

Émission de messages

L'émetteur (cf. figure 14.5) permet au simulateur de traiter les messages de manière asynchrone (de ne pas se bloquer lors d'un envoi de messages) si le routeur n'est pas disponible en faisant office de mémoire entre eux.

Selon les protocoles de communication et le support d'exécution parallèle disponibles sur le processeur (machine) cible, l'émetteur peut être un ensemble de processus en fonction du nombre de canaux de sortie du simulateur élémentaire (PL) pour s'adapter au mieux à la disponibilité du routeur.

L'émetteur (cf. figure 14.4) envoie ses messages vers le système de routage car le problème de mémorisation des messages est supposé résolu.

Dans le cas de la simulation optimiste, l'émetteur mémorise son antimessage à l'émission de chaque message. Ils servent à rétablir la synchronisation lorsqu'une erreur de causalité survient.

Réception de messages

Le récepteur joue également le rôle de mémoire entre le routeur et l'entrée du simulateur élémentaire. Selon le protocole de communication et le support d'exécution parallèle, le récepteur peut se décomposer en plusieurs processus en fonction des canaux d'entrée pour être optimal.

Dans le cas de la simulation optimiste, le récepteur entretient une liste triée des messages d'entrée *inputQueue*. Cette liste est composée de deux sous-listes: la liste triée des messages déjà traités *processedIQ* et la liste triée des messages à traiter *unprocessedIQ*.

L'*inputQueue* comporte la liste triée des messages provenant des différents prédécesseurs.

Remarque 18
Les opérations de réception et de traitement d'une part, et d'émission et de filtre d'autre part, se font directement sur une simulation élémentaire tout-partie (toute la structure en bloc).

La structure récepteur-traitement-émetteur est simple à mettre en œuvre tant que le simulateur élémentaire ne fait que recevoir des messages via le récepteur et en envoyer via l'émetteur. Lorsque le nœud simulé exige une forte synchronisation avec ses prédécesseurs et ses successeurs (échanges de messages dans les deux sens), cette structure devient trop rigide et mal adaptée.

14.3 Analyse quantitative

Cette section montre qu'il est possible de dégager, à partir des observations élémentaires effectuées sur un système, un ensemble de critères de performance tels que la charge du système, le temps de réponse, le taux d'occupation des stations-service. Plusieurs relations fondamentales seront établies entre ces différents critères de performance, quels que soient le système de files d'attente et la période de mesure, à condition de respecter certaines

contraintes. Nous verrons ensuite que ces relations se retrouvent sous des hypothèses probabilistes dans la théorie des files d'attente.

14.3.1 Système élémentaire

Un système est vu comme une boîte noire (cf. figure 14.6) qui reçoit des clients en entrée et les restitue à la sortie après un traitement d'un certain temps. Aucune hypothèse n'est émise quant au comportement de la boîte noire. L'évolution du système est suivie par l'intermédiaire de deux compteurs qui comptabilisent les entrées et les sorties des clients et permettent ainsi de totaliser à chaque instant le nombre de clients en cours de traitement.

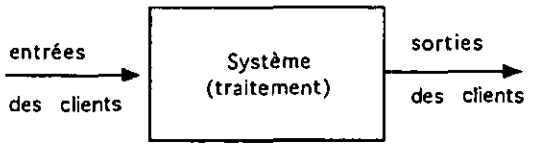


Figure 14.6: Structure d'un système élémentaire

Les mesures élémentaires retenues sont les suivantes:

X: durée de la mesure,

A: nombre total de clients à l'entrée du système,

D: nombre total de clients à la sortie du système,

$X(n)$: durée accumulée pendant laquelle le système contient n clients,

$nmax$: nombre maximum de clients dans le système.

A partir de ces mesures élémentaires, nous pouvons dégager les critères de performances suivants:

λ: charge globale du système (à la sortie),

N: nombre moyen de clients dans le système,

T: temps moyen de réponse du système,

qui sont calculés par les relations:

$$\lambda = D/X$$

$$N = \sum_{i=1}^{nmax} n.X(n)/X$$

$$T = \sum_{i=1}^{nmax} n.X(n)/D$$

De ces relations nous obtenons la formule de Little opérationnelle: $N = \lambda\, T$.

L'influence des clients non terminés à l'issue de la mesure devient négligeable dès que la durée de celle-ci est choisie suffisamment grande. L'équilibre opérationnel du système conduit à une observation telle que $A \cong D$.

14.3.2 Système composé

Considérons un système (cf. figure 14.7) constitué de m sous-systèmes indépendants.

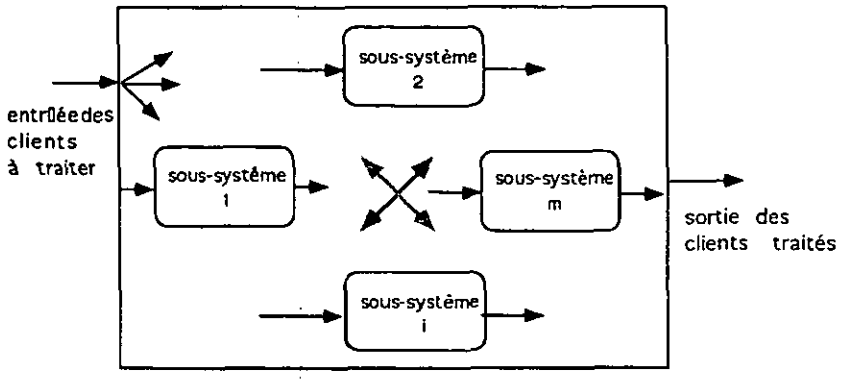


Figure 14.7: Structure d'un système composé

Nous allons montrer que nous pouvons obtenir les critères de performance précédents aussi bien pour chacun des sous-systèmes que pour le système global.

Les mesures élémentaires effectuées sont indicées par i, numéro du sous-système, $i = 1, \ldots, m$:

X: durée de la mesure,

D: nombre total de clients sortis du système,

D_i: nombre total de clients traités par le sous-système i,

$X_i(n)$: durée cumulée pendant laquelle le sous-système i contenait n clients élémentaires.

Les critères de performance sont enrichis par:

λ_i: charge locale de la station i,

U_i: taux d'occupation de la station i (proportion du temps pendant lequel la station i est occupée),

S_i: durée moyenne de traitement d'un client (service) dans le sous-système i,

e_i: nombre moyen de visites au sous-système i par travail,

T_i: temps moyen de réponse du sous-système i,

N_i: nombre moyen de clients dans le sous-système i,

λ: charge globale du système.

Les critères précédents sont calculés conformément aux relations ci-après:

$$\lambda_i = D_i/X$$

$$U_i = (X - X_i(0))/X$$

$$S_i = (X - X_i(0))/D_i$$

$$e_i = D_i/D$$

$$R_i = \sum_{j=1}^{nmax} j.X_i(j)/D_i$$

$$L_i = \sum_{i}(j)/X$$

$$\lambda = D/T$$

Nous en déduisons la version quantitative du théorème de Chang-Lavenberg qui relie la charge globale du système aux charges locales de chacun des sous-systèmes i:

$$\lambda = \lambda_i/e_i = U_i/S_i e_i$$

$$\lambda = N_i/e_i T_i \Rightarrow \lambda_i = N_i/T_i$$

Nous retrouvons ici l'expression de la relation de Little "locale" (appliquée à chaque sous-système). Nous allons par aillenrs vérifier sa validité au niveau global en calculant les grandeurs:

N: nombre moyen de clients dans le système,

T: temps moyen de réponse dans le système.

Sous l'hypothèse qu'un travail n'engendre qu'un seul client, nous avons:

$$N = \sum_{j=1}^{m} N_i$$

$$T = \sum_{j=1}^{m} e_i T_i$$

et enfin: $N = \lambda T$

14.4 Spécification des nœuds et leur analyse

L'environnement de programmation SEDMA utilisé pour tester les différents réseaux de simulation a été développé en parallèle avec Olivier von Dach dans le cadre de son travail de diplôme.

Les différents types de nœuds mis à disposition par SEDMA, pour simuler un réseau de file d'attente sont: le nœud source ("source node"), le nœud branchement ("branch node"), le nœud station-service ("servers node") et le nœud puits ("sink node"). Les concepts généraux définis ci-dessus sont applicables à tout nœud du réseau simulé de la file d'attente. Nous allons étudier comment spécifier chaque type de nœud et quelles sont les variables statistiques qu'il faut retenir en chaque nœud pour étudier la performance du système de la file d'attente.

14.4.1 Nœud source

Il ne possède pas de canal d'entrée mais uniquement des canaux de sortie. Le comportement d'un nœud source est défini en spécifiant le mécanisme des arrivées (la loi de répartition des interarrivées) et éventuellement la période de simulation, le nombre de clients à traiter ainsi que les différents types (classes) de clients à engendrer. La spécification se fait à l'aide du code Smalltalk qui par la suite est compilé et intégré au moteur de simulateur dynamique optimiste.

Étude statistique

Les données statistiques qui peuvent être manipulées (examinées et modifiées), à tout instant, sont:

- le nombre de clients engendrés, par type, et la somme totale de clients créés;
- le temps moyen des interarrivées;

14.4.2 Nœud branchement

Un nœud branchement possède au moins deux canaux de sortie et au moins un canal d'entrée. Il possède également des "poids" ou la probabilité pour emprunter un canal donné de sortie. Le critère de branchement sur chaque canal est établi à l'aide du code Smalltalk.

Étude statistique

Le nombre de clients qui emprunte chaque branchement et ceux qui arrivent sans critère de branchement.

14.4.3 Nœud station-service

Une station-service se compose d'éléments essentiels suivants:

- une file d'attente;
- un ou plusieurs serveurs attachés à cette file d'attente fournissant des prestations équivalentes;

- la description algorithmique du service, fournie à l'aide du code Smalltalk;
- un algorithme de routage qui permet de définir la circulation des clients dans le réseau.

Les clients du réseau sont créés à un nœud source, circulent entre les stations-services où ils reçoivent un service et entre les nœuds de branchement où ils empruntent le bon itinéraire pour finir leur parcours (activité) dans un nœud puits. Il est possible de caractériser des clients en leur attribuant des classes distinctes.

Les clients se présentent à l'entrée de la station-service et sont aiguillés sur un serveur libre. Si tous les serveurs sont occupés, alors ils sont mis dans une file d'attente unique. Lorsqu'un serveur est libéré, le client, en tête de la file d'attente, est pris en charge par le serveur. Si la file d'attente est vide, le serveur devient inactif et libre. Les données supposées connues sont: la loi d'arrivée des clients à la station-service et la loi de répartition des temps de service de chaque serveur.

Etude statistique

Elle se compose de l'étude statistique relative à une file d'attente et aux serveurs. Les données statistiques intervenant dans l'analyse d'une file d'attente sont:

- la taille courante de la file d'attente;
- la longueur moyenne de la file d'attente;
- la longueur maximale de la file d'attente;
- le temps moyen d'attente;
- le temps critique (maximal) d'attente;

Les données statistiques intervenant dans l'analyse de la performance d'un serveur sont:

- le temps moyen de service,
- le temps maximum de service,
- le nombre de clients traités,
- son état courant ou instantané (occupé ou libre),
- son taux d'occupation.

Les données statistiques relatives à une station-service (groupe de serveurs) sont:

- le nombre moyen de serveurs en service,

- le nombre de clients servis par la station-service,
- le taux d'occupation des serveurs de la station-service,
- le temps critique (maximum) de service,
- la charge de la station-service,
- le temps d'attente moyen et maximum (critique).

14.4.4 Nœud puits

Il sert à éliminer un client du réseau simulé. Il ne correspond à aucun nœud du système réel simulé. Les données statistiques attachées au nœud puits sont:

- le nombre de clients traités dans le système,
- le temps moyen passé dans le système,
- la charge du système,
- le temps moyen d'attente dans le système.

14.5 Etude de la performance

L'étude de la performance comporte deux parties. La première traite de deux modèles simples (paragraphe 2) mettant en valeur la convivialité de l'interface. La seconde traite des modèles théoriques complexes de réseaux de file d'attente (paragraphe 3) . Chaque cadre d'expérience est expliqué (paragraphe 1). Les expériences sont exécutées sur Sun dans le langage de programmation Smalltalk80. Les autres paragraphes présentent la synthèse et les leçons tirées des expérimentations sur les algorithmes décrits.

14.5.1 Cadre d'expérimentation

Les réseaux de file d'attente simulés sont alimentés par un nombre fixe de clients (ou par une période de simulation donnée). Un client entre dans le système par un nœud source, attend un service à une station de service, est servi et sort pour entrer dans une autre station de service puis termine son parcours dans un nœud puits. Chaque nœud possède canaux d'entrée (C_{in}) et canaux de sortie (C_{out}). La discipline de la gestion des canaux est PAPS. Le service est non-préemptif. L'interarrivée et le temps de service suivent une loi de répartition que nous spécifions pour le modèle simulé.

Algorithmes testés

Un algorithme de simulation optimiste décrit une combinaison des mécanismes de contrôle local et de contrôle global. Selon la combinaison, nous distinguons, d'après la figure 14.5.1, quatre types d'algorithmes décrits dans la thèse:

- algorithme optimiste standard;
- algorithme adaptatif;
- algorithme LT de la date locale sûre;
- algorithme combinant l'algorithme adaptatif et l'algorithme LT.

	local standard	local adaptatif
GVT	optimiste	algorithme adaptatif
GVT local ou LT sûr	algorithme LT-sûr	algorithme adaptatif et LT-sûr

Figure 14.8: Différents algorithmes de la thèse

L'étude de la performance de l'algorithme adaptatif consiste à déterminer la taille de la place mémoire nécessaire pour la simulation. La taille maximale de la liste des états sauvegardés donne l'estimation de la place mémoire nécessaire pour mener à bien une expérience de simulation.

Remarque 19
On pourrait pousser plus loin l'étude de la qualité de l'algorithme en évaluant le modèle adaptatif selon les critères suivants: le temps d'exécution, le nombre de rollbacks et la longueur moyenne des états sauvegardés.

Logiciel de simulation

La simulation est réalisée sur la plateforme interactive de la simulation par événements discrets multi-agents ou SEDMA. Cette plateforme est conçue pour expérimenter les algorithmes décrits dans cette thèse. Nous l'utilisons dans le cadre particulier de la simulation des réseaux de files d'attente. Comme toute application répartie, les performances de la simulation dépendent fortement des caractéristiques du modèle simulé et de l'architecture sur laquelle la simulation est exécutée.

14.5.2 Description des modèles simples

Système de lavage de voitures

Des voitures à laver arrivent à une station-service (disposant par exemple de deux serveurs). Si les serveurs sont occupés, les voitures entrent dans une liste d'attente. Si un serveur est libre, la voiture se dirige vers celui-ci pour être lavée. La discipline de service est PAPS, le temps de service et l'arrivée des voitures à la station sont connus. On se

propose de déterminer le nombre moyen de stations à mettre à disposition pour que les voitures à laver attendent pendant un intervalle de temps raisonnable. La figure 14.9 présente la structure du réseau simulé associée au système de lavage de voitures. Les résultats sont obtenus à l'aide de l'algorithme optimiste standard.

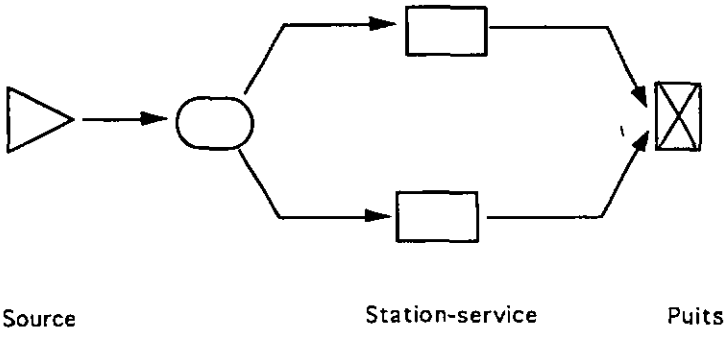


Figure 14.9: Réseau simulé du système de lavage de voitures

L'interface pour la spécification du système de lavage de voitures est présentée dans l'annexe B. L'interface graphique pour l'analyse des données statistiques du système de lavage de voitures est présentée dans l'annexe C.

Comptoir CFF

Prenons un exemple où le passage du système étudié au modèle de files d'attente est assez simple à réaliser. Nous étudions, pour le compte des CFF (version adaptée de Fdida [PF89]), le comportement des usagers devant les guichets où sont délivrés les billets et les abonnements.

Notre première tâche consiste à identifier les divers éléments à prendre en considération:

- les **clients ou usagers**: la gare qui sert de prétexte à notre étude reçoit à peu prés un usager toutes les 20 secondes. Les usagers modélisent les clients ou les composants passifs du système.

- Les **guichets ou station-service**: le personnel actuellement disponible permet d'ouvrir cinq guichets. Un usager met 1 mn 30 s pour être servi quel que soit le guichet. Les guichets modélisent les centres d'activité. Il y a une file d'attente devant chaque guichet. Nous supposons qu'un usager se dirige de manière équiprobable vers l'un des cinq guichets disponibles sans se préoccuper de l'état de la file d'attente devant chacun des guichets.

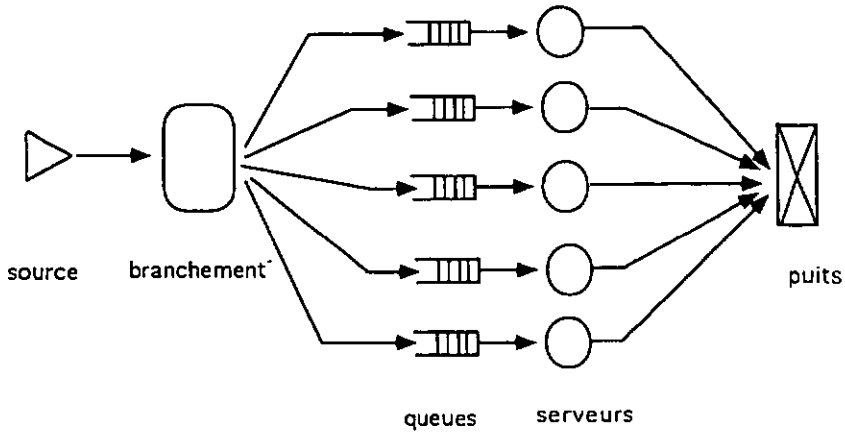


Figure 14.10: Cheminement statique et dynamique des clients

La structure du réseau simulé est donnée par la figure 14.10.

L'activité d'un usager (le parcours complet) consiste à se diriger vers un guichet où il obtiendra le service demandé, puis à se diriger vers la sortie du système.

14.5.3 . Description des modèles complexes

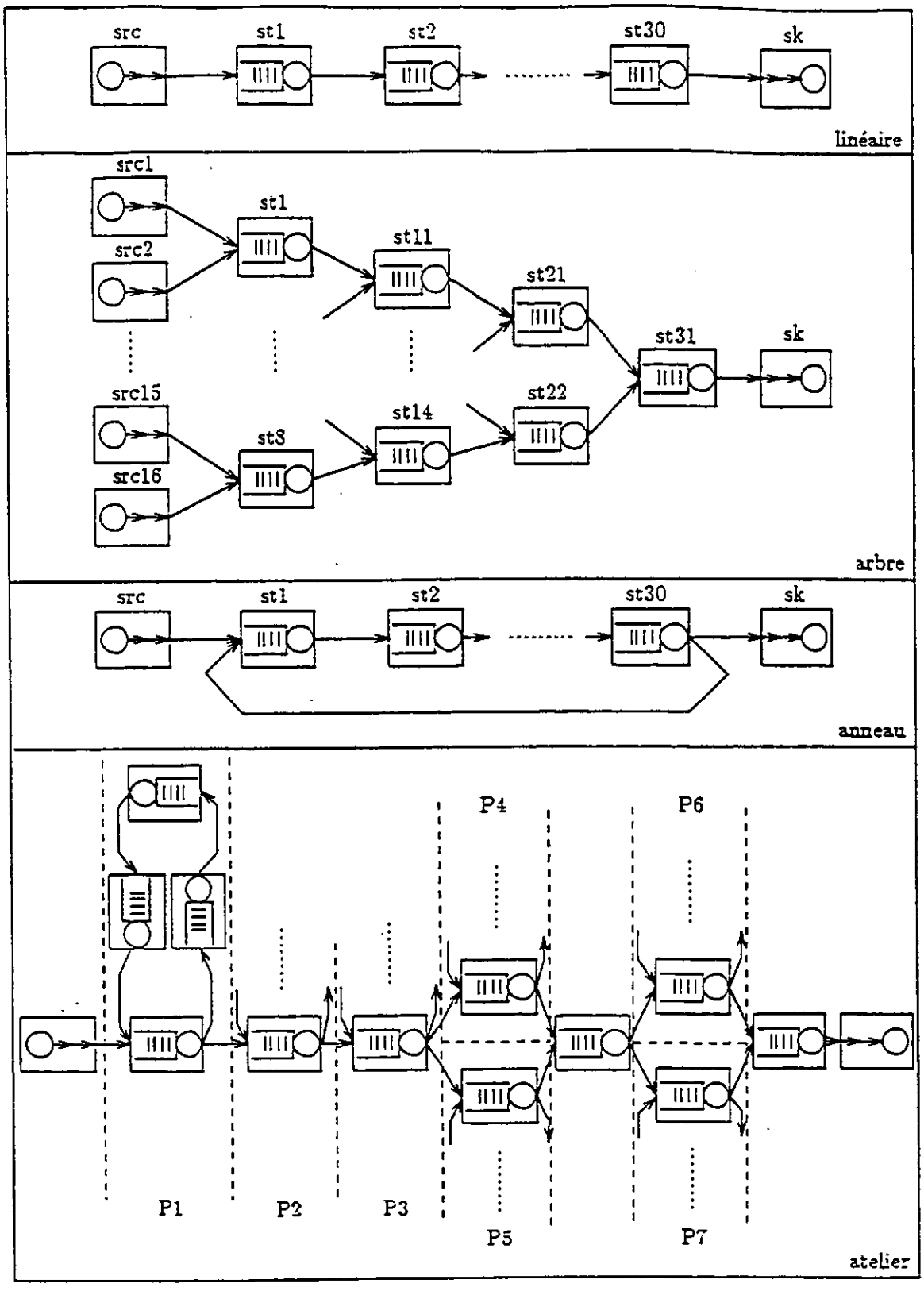


Figure 14.11: Topologie des modèles complexes simulés

Nous allons considérer cinq exemples de réseaux de file d'attente pour évaluer les performances des algorithmes décrits dans cette thèse. Ces exemples illustrent cinq topologies de modèle simulé. La figure 14.11 montre quatre topologies de modèle: la première est un ensemble de stations-service disposées en ligne; la deuxième en arbre, la troisième en anneau et la quatrième est un atelier de production combinant les trois topologies précédentes.

Le premier modèle linéaire de la figure 14.11 est constitué d'une source, de 30 stations de service et d'un puits. Le deuxième est un arbre binaire de 16 feuilles qui sont les sources, la racine étant le puits; les autres nœuds de l'arbre sont constitués de 15 stations-service. Le troisième est un anneau constitué de 30 stations-service, d'une source pour introduire les clients dans le modèle et d'un puits. Le quatrième est un atelier de production constitué d'une source, d'un puits et de 7 sous-ateliers $P1, \ldots, P7$.

Les sources considérées sont caractérisées par une loi d'interarrivée exponentielle de moyenne 2.0 unités et le temps de service des serveurs, la loi normale de 2.0 unités de temps avec une variance de 0.5.

Topologie du modèle diffusion-fusion

Afin de bien relever l'influence de la topologie du modèle sur le calcul de la date locale sûre, nous considérons le modèle (cf. figure 14.12). Le modèle est constitué d'une source, d'un dispatcher, de 10 stations-service et d'un puits. C'est le cas typique où l'algorithme de GVT est le moins performant. Ce canal a une probabilité presque nulle (ici 1 pour 1000) d'être choisi pour la suite d'un message d'événement.

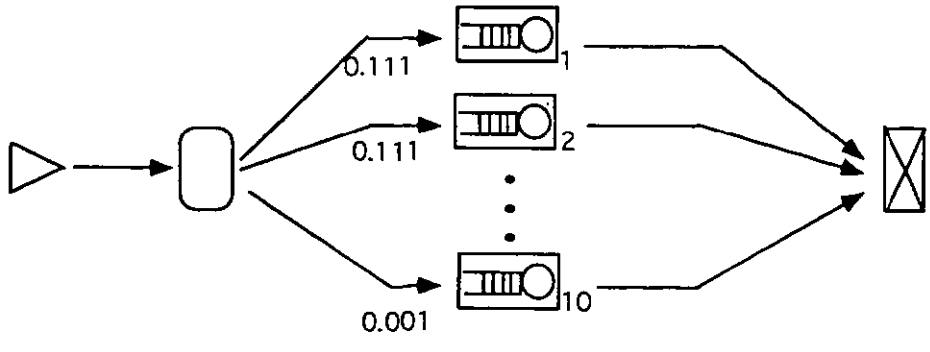


Figure 14.12: Topologie du modèle diffusion-fusion

14.5.4 Analyse des résultats

Les résultats sont représentés par type de topologie. Les résultats de l'algorithme de référence sont ceux de la simulation optimiste standard (algorithme GVT).

Topologie du modèle linéaire

Les figures (figure 14.13, figure 14.14 et figure 14.15) présentent les résultats des trois premiers types d'algorithmes décrits. Les résultats du quatrième algorithme sont sensiblement identiques à ceux de l'algorithme adaptatif. C'est la raison pour laquelle sa figure n'est pas présentée.

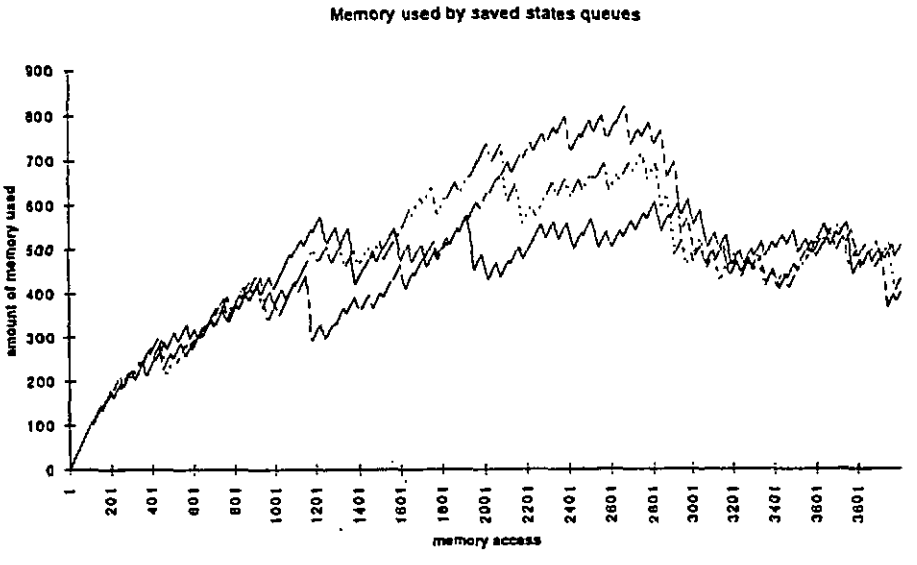


Figure 14.13: performance de l'algorithme optimiste - linéaire

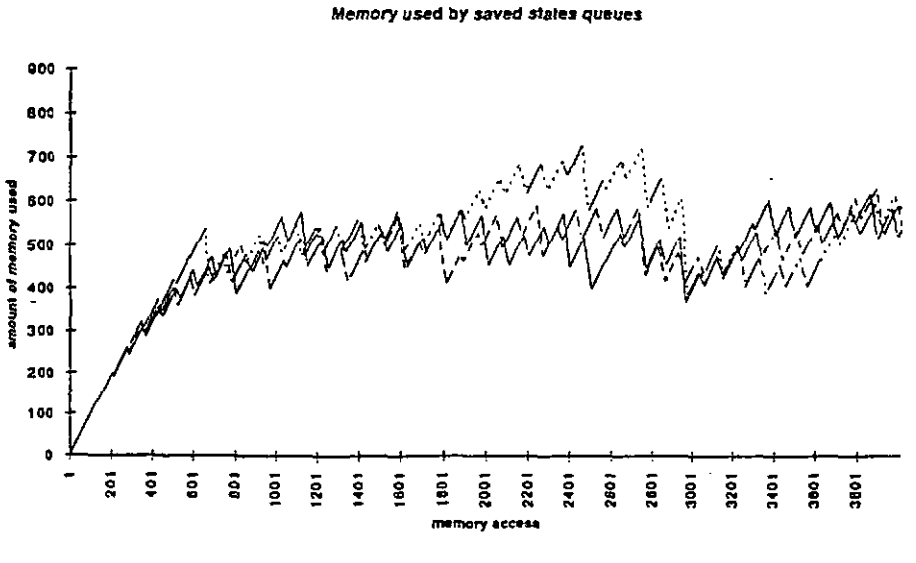


Figure 14.14: Performance de l'algorithme LT - linéaire

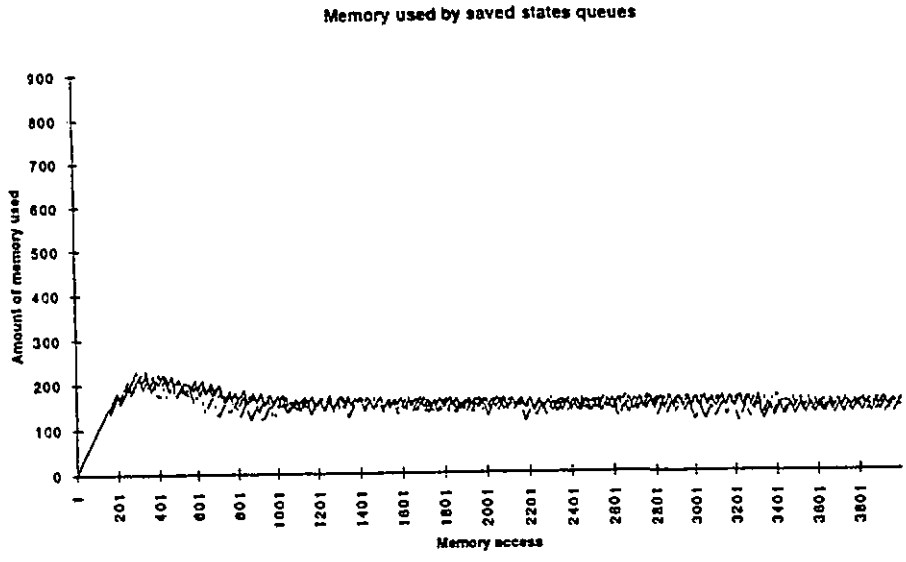


Figure 14.15: Performance de l'algorithme adaptatif - linéaire

Nous constatons que la consommation moyenne de mémoire pour l'algorithme optimiste (cf. figure 14.13) varie entre 200 et 800. La même expérience en utilisant l'algorithme LT (cf. figure 14.14) montre que la consommation de la mémoire varie en moyenne entre 350 et 700. L'expérience répétée en utilisant l'algorithme adaptatif dont la fenêtre temporelle varie entre 5 et 20 montre que la consommation de la mémoire (cf. figure 14.15) est en moyenne entre 100 et 200. L'algorithme auquel sont intégrés l'algorithme LT et l'algorithme adaptatif ne fournit pas un résultat meilleur que l'algorithme adaptatif.

Remarque 20
Nous préférons définir les bornes de la fenêtre temporelle par des nombres $n1$ et $n2$ que par le temps $t1$ et $t2$. Les nombres $n1$ et $n2$ désignent le nombre d'états dans la liste des états sauvegardés qui peuvent varier entre $n1$ et $n2$.

Topologie du modèle en arbre

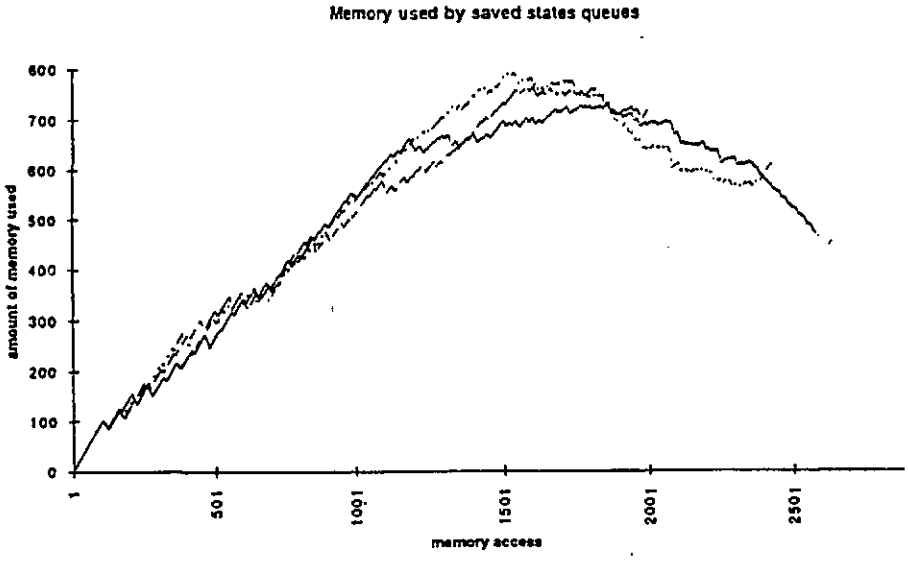


Figure 14.16: Performance de l'algorithme optimiste - arbre

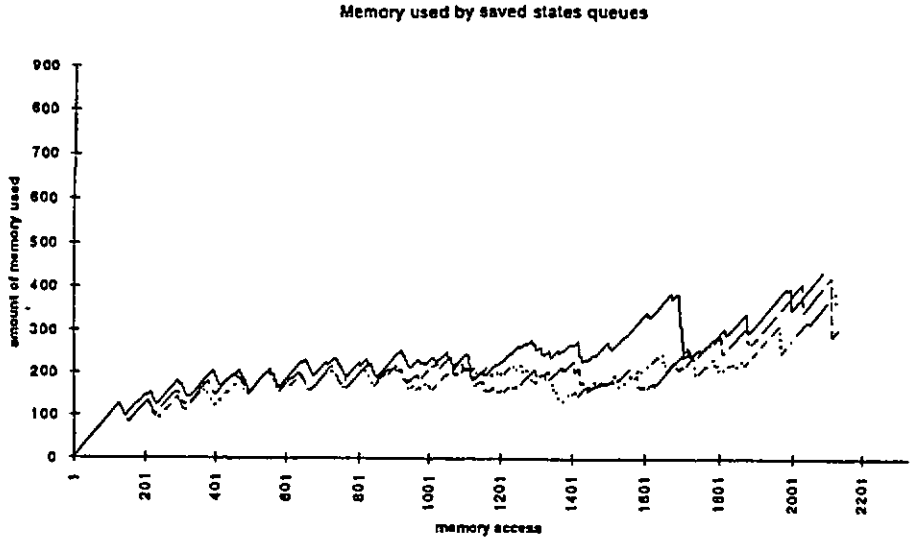


Figure 14.17: Performance de l'algorithme LT - arbre

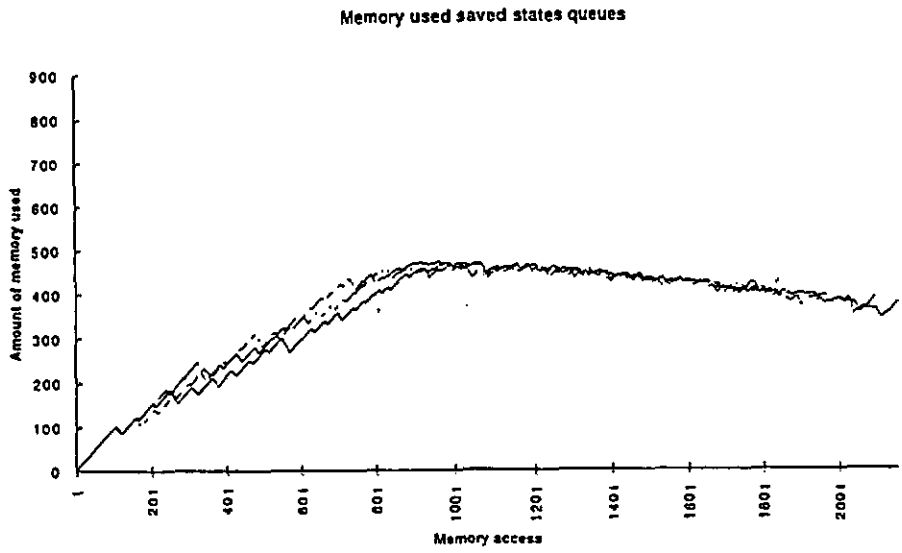


Figure 14.18: Performance de l'algorithme adaptatif - arbre

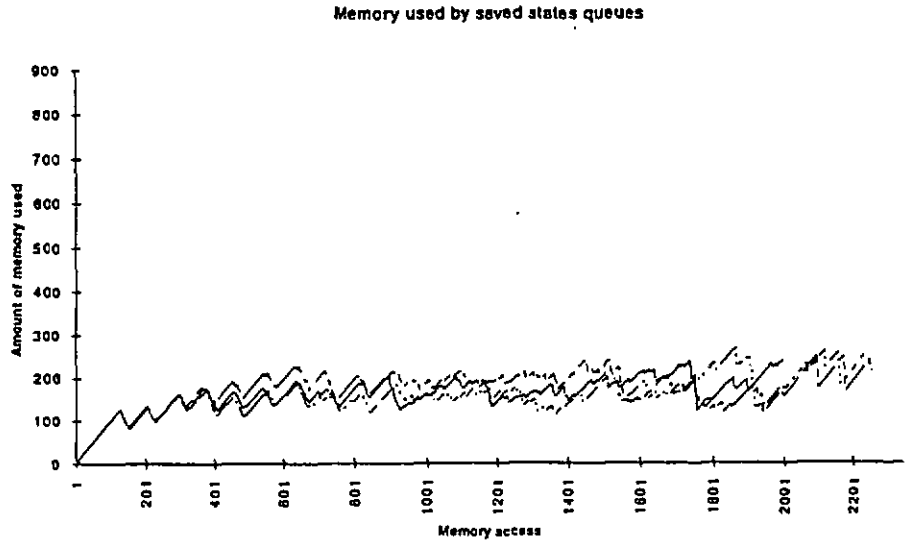


Figure 14.19: Performance de l'algorithme LT et adaptatif - arbre

Ces expériences montrent que la meilleure performance s'obtient lorsqu'on combine l'algorithme LT et l'algorithme adaptatif (cf. figure 14.19) dont la taille de la fenêtre temporelle varie entre 15 et 30. La seconde meilleure performance est celle de l'algorithme LT seul (cf. figure 14.17); sa consommation de mémoire est en moyenne le double par rapport à la meilleure performance (cf. figure 14.19). La troisième est celle de l'algorithme adaptatif seul (cf. figure 14.18), où le facteur de consommation de la mémoire devient 4 fois plus grande que la meilleure. Ensuite vient l'algorithme optimiste standard, où le facteur de consommation de la mémoire par rapport à la meilleure s'élève à 8 fois plus grande. Ce dernier algorithme est le moins bon dans un modèle en arbre.

Topologie du modèle en anneau

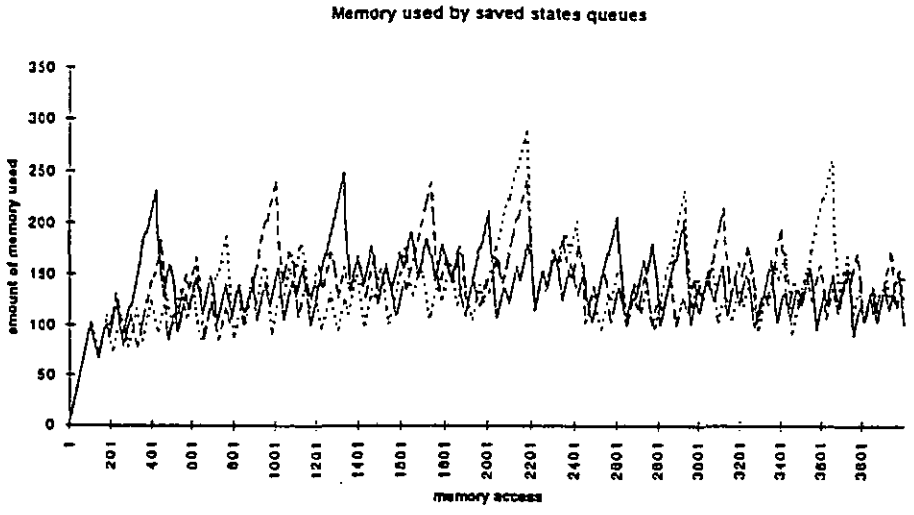


Figure 14.20: Performance de l'algorithme optimiste pour l'anneau

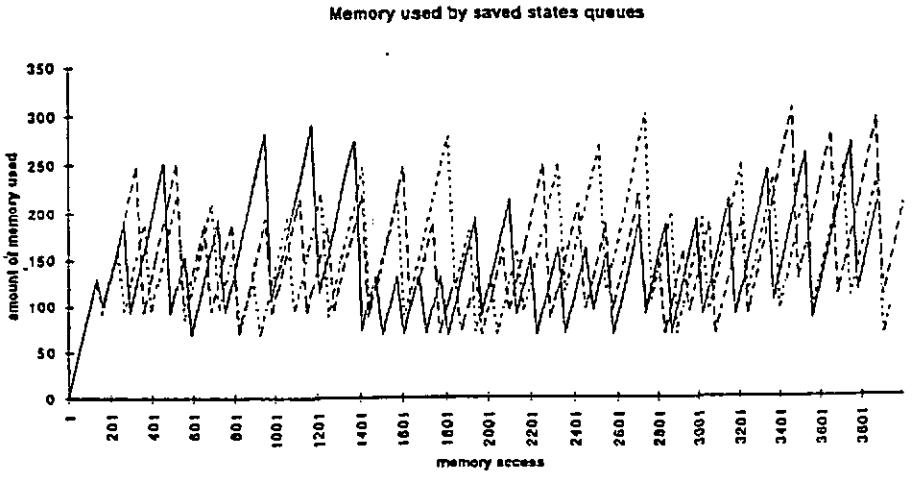


Figure 14.21: Performance de l'algorithme LT pour l'anneau

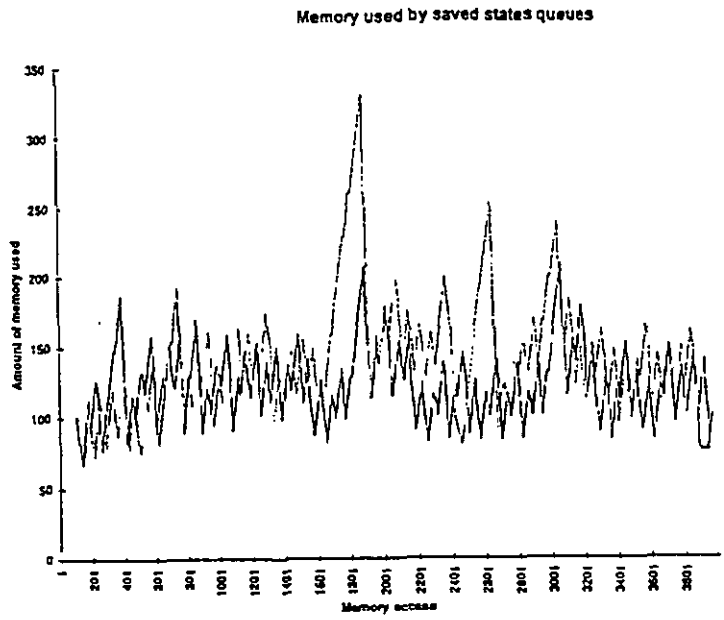


Figure 14.22: Performance de l'algorithme adaptatif pour l'anneau

Ces expériences (figure 14.20, figure 14.21 et figure 14.22) montrent que pour la topologie en anneau ni l'algorithme LT ni l'algorithme adaptatif n'apportent un gain de consommation de mémoire. Il est cependant intéressant de remarquer que la performance des algorithmes proposés (cf. figure 14.21 ou cf. figure 14.22) n'altère pas celle de l'algorithme optimiste standard (cf. figure 14.20).

Topologie du modèle en atelier de production

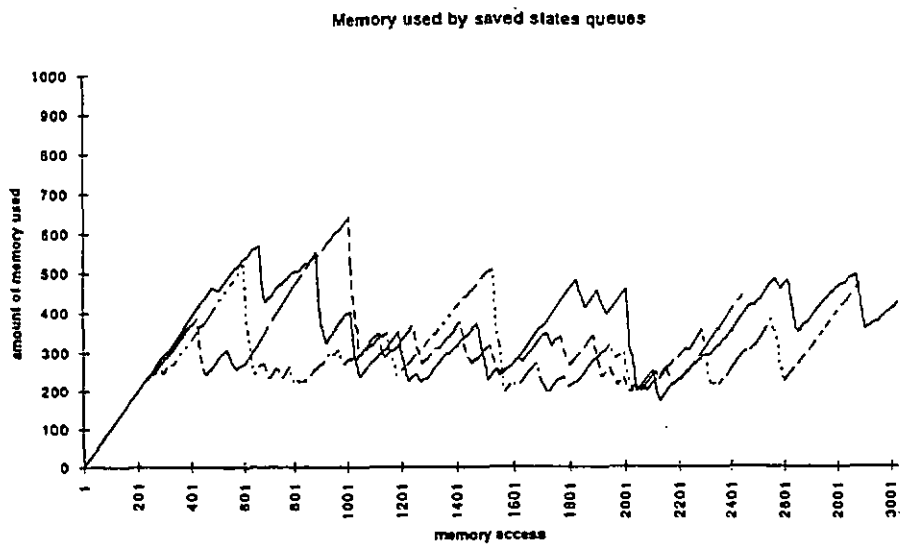


Figure 14.23: Performance de l'algorithme optimiste - atelier de production

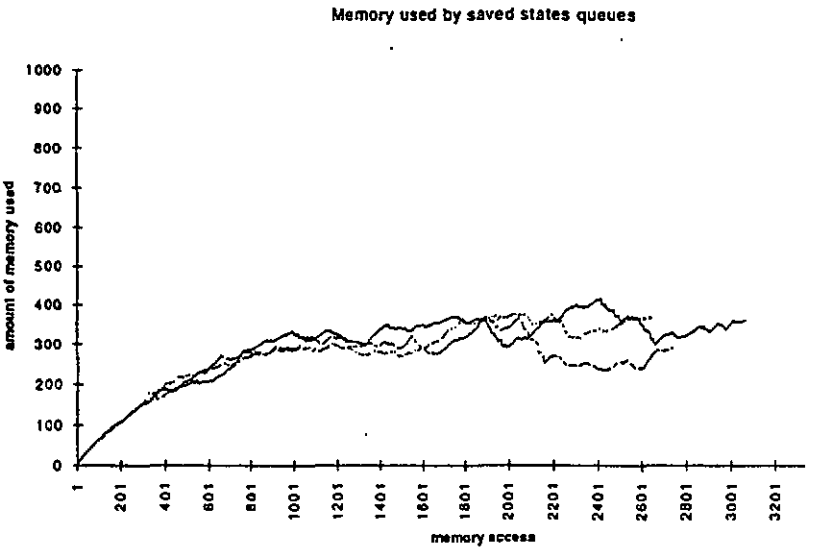


Figure 14.24: Performance de l'algorithme LT - atelier de production

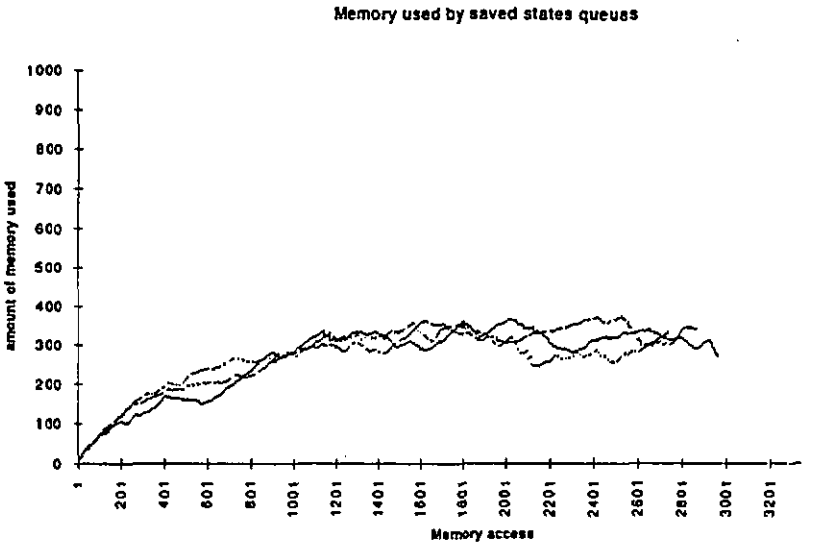


Figure 14.25: Performance de l'algorithme adaptatif - atelier de production

Ces expériences (figure 14.23, figure 14.24 et figure 14.25) montrent que la consommation de la mémoire de l'algorithme adaptatif, dont la taille de la fenêtre temporelle varie entre 10 et 30, est presque la même que celle de l'algorithme LT (figure 14.24). Par contre, la figure 14.23 montre que le résultat de la consommation de l'algorithme optimiste est moins performant que ceux des algorithmes proposés. L'amélioration est de facteur 2 par rapport à l'algorithme adaptatif (cf. figure 14.24), qui lui, fournit la meilleure performance.

Topologie du modèle diffusion-fusion

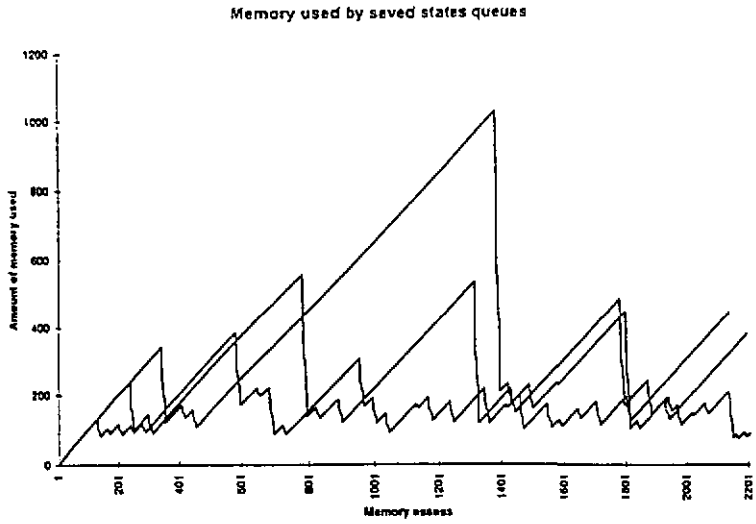


Figure 14.26: Performance de l'algorithme adaptatif - diffusion-fusion

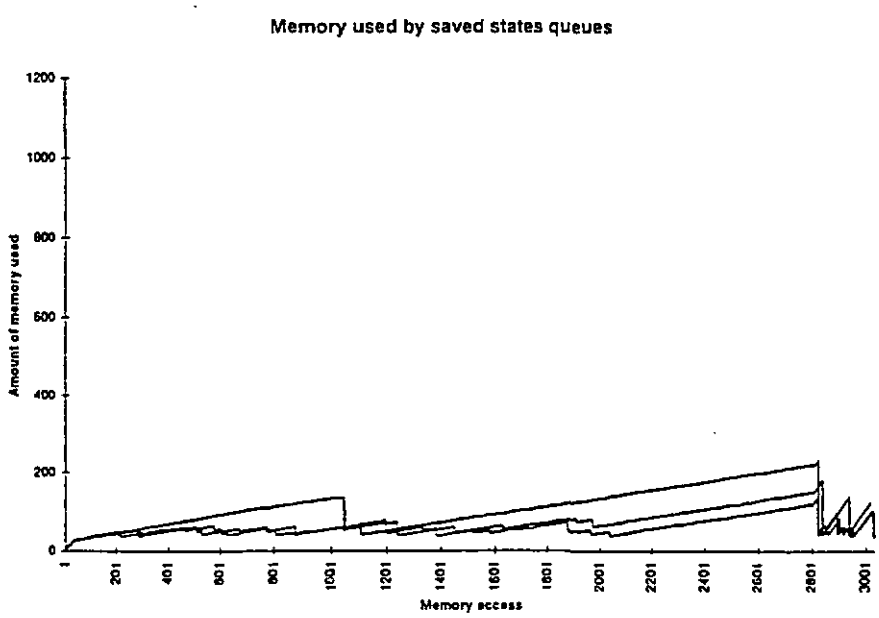


Figure 14.27: Performance de l'algorithme LT - diffusion-fusion

Cet exemple sert à évaluer uniquement la performance de l'algorithme LT, là où tous les algorithmes de GVT sont moins performants. Il révèle également la qualité de la performance de l'algorithme de la date locale sûre.

Ces expériences (figure 14.26, figure 14.27) montrent que la consommation de la mémoire de l'algorithme GVT par rapport à l'algorithme LT est 6 fois plus grande. Autrement dit, pour simuler le même modèle, l'algorithme de GVT standard consomme six fois plus de mémoire que l'algorithme LT de la date locale sûre.

14.5.5 Synthèse des résultats selon les différentes topologies

Les performances de SEDMA se dégradent au fur et à mesure que le modèle simulé est constitué d'une grande composante fortement connexe englobant un grand ensemble de nœuds. Bien que l'augmentation de la connexité du modèle dégrade les performances, celles de la topologie en anneau sont parfaitement comparables à celles de l'algorithme de GVT. Ceci est dû au fait que le calcul de la date locale sûre fait intervenir l'ensemble des nœuds du système à l'instar de l'algorithme de GVT. Mais la différence fondamentale entre ces deux algorithmes est que l'algorithme LT n'a besoin ni de bloquer tous les processus logiques ni d'un gestionnaire central pour aboutir au même résultat.

Les aspects de la topologie du modèle qui influencent la performance de l'algorithme LT sont:

- le nombre de prédécesseurs des stations de service (nœuds) dans un modèle influence les courtes périodes de synchronisation du jeton entre les PLs d'une même composante fortement connexe. C'est la raison pour laquelle la topologie en arbre fournit la meilleure performance;
- le nombre de PLs dans une boucle (un circuit) dans le modèle définit la dépendance d'une station de service (d'un PL) par rapport aux autres PLs de la boucle. La quantité de jetons de synchronisation peut être très importante selon les paramètres des stations-service impliqués dans un circuit et dégrade fortement les performances. C'est le cas de la topologie en anneau qui fournit la moins bonne performance;
- lorsque dans la topologie du modèle, l'émission des messages est déséquilibrée comme le nœud de branchement dans la topologie diffusion-fusion, la performance du modèle se révèle encore meilleure par rapport à l'algorithme de calcul de GVT.

Coût de communication des jetons

Le coût de communication des jetons, bien que moins important par rapport aux calculs liés aux messages d'événement, n'est pas nul. Le traitement des jetons introduit des coûts temporels d'estimation de la date locale sûre. Ces coûts correspondent aux périodes durant lesquelles les PLs (ou AgSED) ne s'occupent pas du traitement des messages d'événement. La faible synchronisation des jetons sur les canaux d'entrée d'un PL (ou d'une composante fortement connexe) peut conduire à des courtes périodes d'attente de jeton sur les canaux d'entrée.

Synthèse de l'algorithme adaptatif

L'algorithme adaptatif fournit une meilleure performance pour un modèle linéaire car la fenêtre temporelle permet de mieux répartir le flux des messages d'événement de la source vers la sortie. Ceci d'autant que le modèle ne possède aucun nœud de synchronisation: la source termine vite son activité et crée le goulot d'étranglement au début de la simulation au niveau des nœuds en aval, ce qui explique leur mauvaise performance.

Cet algorithme nécessite un choix judicieux des bornes de la fenêtre temporelle. En effet, pour une fenêtre temporelle de petite taille (de l'ordre de dizaine), on termine la simulation par manque de mémoire. Pour une fenêtre temporelle de grande taille, la performance qui en résulte est moindre par rapport à l'algorithme optimiste standard. Les meilleures bornes sont liées au type de problème que l'on veut résoudre.

Synthèse de l'algorithme LT

On constate que SEDMA fournit:

- la meilleure performance pour une topologie en structure d'arbre symétrique et équilibré;
- la moins bonne performance pour une topologie en anneau.

La performance de l'algorithme LT dépend de la politique d'optimisation d'envoi des messages de jetons. Plusieurs possibilités sont envisageables; en voici quelques-unes:

- envoi de jeton après un délai donné de chaque source;
- envoi de jeton après n traitements de messages d'événement;
- seules les sources engendrent des jetons et les autres nœuds (PLs/ AgSED) ne font que les envoyer plus loin jusqu'aux sorties. On pourrait admettre que les serveurs engendrent aussi des jetons après k traitements de messages d'événement (ou après un certain délai), particulièrement en présence d'une topologie en anneau pour la date locale sûre à jour.

Chapitre 15

Conclusion générale

Introduction

La mise en œuvre de la distorsion temporelle comporte deux parties majeures: le mécanisme de contrôle local qui assure le respect de la contrainte de causalité locale et le mécanisme de contrôle global qui traite des questions globales telles que la détection de la terminaison, la validation de l'ensemble du simulateur jusqu'à une date donnée, la gestion de mémoire et la gestion des entrées-sorties.

La contribution dans la mise en œuvre de la distorsion temporelle s'inscrit dans deux directions orthogonales:

- rendre efficace le contrôle local en modifiant dynamiquement les paramètres de contrôle de chaque PL;
- proposer un nouvel algorithme LT sûr qui substitue la fonctionnalité d'un algorithme estimant GVT. Le principe de cet algorithme consiste à définir une estimation de la date locale sûre, pour chaque PL, qui valide les opérations qu'aurait validées une estimation de GVT.

A ces deux contributions majeures s'en ajoute une moins importante qui consiste à doter le simulateur d'une interface graphique interactive de simulation:

1. pour décrire le modèle à simuler à l'aide d'un graphe de PLs;
2. pour suivre les résultats partiels de la simulation où l'utilisateur est un participant actif, capable de lancer, de terminer et de suspendre l'exécution d'un PL afin d'examiner son état ou en modifier partiellement l'ensemble des variables;
3. pour produire le résultat d'une expérience de simulation dans un fichier ou dans une fenêtre écran.

Ce chapitre est structuré en trois sections: les deux premières présentent notre contribution à la mise en œuvre de la simulation optimiste; la dernière présente les axes de recherche à développer.

15.1 Contribution au contrôle local

Un nouveau concept basé sur la "qualité de synchronisation" $Q_{synchro}$ a été introduit pour chaque PL dans une simulation répartie optimiste. Les contributions du mécanisme dynamique de la distorsion temporelle consistent à:

- réduire la place mémoire utilisée en modifiant dynamiquement, selon la valeur de $Q_{synchro}$, la période de sauvegarde d'états de chaque PL;
- équilibrer l'avancement des LVTs du simulateur à des vitesses proches en
  - modifiant dynamiquement, selon la valeur de $Q_{synchro}$, la fenêtre temporelle de chaque PL,
  - ordonnançant dynamiquement, selon la valeur de $Q_{synchro}$, chaque PL sur un processeur.

Hormis ces avantages majeurs, on peut encore retenir:

- l'introduction de la notion de simulateur stable,
- l'établissement des critères d'un ordonnanceur spécifique à la simulation optimiste,
- l'intégration de l'approche dynamique dans la simulation répartie par événements discrets,
- l'intégration de l'approche multi-agents dans la simulation dynamique optimiste pour réduire la place mémoire utilisée, ainsi que pour définir une interface graphique interactive.

15.2 Contribution au contrôle global

Dans le chapitre 10, nous avons décrit un nouvel **algorithme LT** pour déterminer en chaque PL, l'équivalent de GVT. Cet algorithme corrige les défauts majeurs des algorithmes existants de GVT en

- déterminant l'équivalent de GVT sans figer ou geler les PLs;
- déterminant la date locale sûre LT d'un PL sans avoir à examiner tous les PLs qui se trouvent dans le système; on élimine ainsi la présence de l'organe central de contrôle;
- calculant la date locale sûre sans avoir recours à un réseau supplémentaire auparavant, dédié exclusivement au calcul de GVT.

L'avantage de l'algorithme LT réside dans le fait que les différents PLs peuvent avoir différentes valeurs de la date locale sûre LT. Rappelons que GVT est dans ce cas la plus petite valeur des LTs. Les PLs dont la valeur de LT est ultérieure à GVT sont plus précis

et utilisent moins de place mémoire et ceux, dont la valeur LT est égale à celle de GVT utilisent la même place mémoire.

En résumé, dans tous les cas, l'algorithme LT est plus précis et utilise moins de mémoire que les algorithmes existants de GVT.

De plus, nous avons établi la preuve formelle de l'exactitude de l'algorithme LT (cf. chapitre 11).

15.3 Perspectives

15.3.1 Répartir le simulateur

La version actuelle de SEDEMA fonctionne dans un environnement réparti simulé. Cette lacune doit être comblée pour tester l'efficacité de l'algorithme d'ordonnancement proposé à l'aide de la qualité de synchronisation. La suite des travaux consistera donc à réaliser un simulateur réparti, puis à intégrer dans le simulateur "un module ou une classe" d'évaluation automatique du parallélisme intrinsèque du modèle simulé et de la quantité de messages sûrs transmis par chaque PL du simulateur. Ces données permettent d'évaluer les accélérations possibles de la simulation et d'optimiser, de manière automatique, le placement des processus logiques sur la configuration matérielle cible. Cela entraîne des problèmes de placement.

Le problème de placement des PLs optimistes (équilibre de charge entre les processeurs et la bonne répartition des communications) doit être étudié pour différentes applications parallèles et réparties [Tal93]. Selon l'application, un placement statique ou un placement dynamique peuvent être mis en œuvre. Le placement dynamique est approprié si les informations nécessaires au placement sont disponibles ou accessibles lors de l'exécution.

15.3.2 Améliorer l'interface

La description d'un modèle à simuler, dans son état actuel, se fait de manière graphique dans l'environnement de simulation: SEDEMA. Cette interface est conçue pour mettre en place un langage général de simulation intégré à Smalltalk. Elle sert à décrire et à visualiser tout type de modèle à simuler, de tous les domaines, à l'aide d'un graphe des PLs du modèle. Elle masque ainsi à l'utilisateur la gestion du parallélisme et le mécanisme de synchronisation des différents composants du modèle.

L'approche "objets" utilisée dans Smalltalk, pour développer l'environnement, fournit des bibliothèques de classe de base pour gérer le mécanisme de synchronisation de la simulation répartie. Il est souhaitable d'étendre les bibliothèques de classes spécifiques pour différents domaines d'application (par exemple les réseaux de files d'attente, les problèmes de la production dans un atelier flexible). L'approche *objet* permet une modularité du simulateur et facilite sa maintenance et son évolution.

15.3.3 Analyse statistique avancée

Les résultats de la simulation étant basés sur des variables aléatoires, il est important de dégager des estimations pour appréhender la précision statistique de manière à obtenir des estimations de précision donnée sur les moyennes déterminées. Etendre l'étude statistique de telle manière que le moteur de SEDMA mette à disposition dans sa couche d'analyse statistique:

- les méthodes analytiques exactes
  - les algorithmes de convolutions,
  - une résolution markovienne,
  - l'analyse des moyennes de valeurs ou MVA ("Mean value analysis");
- les méthodes analytiques approchées
  - les méthodes itératives,
  - les approximations par diffusion,
  - les méthodes heuristiques.

Ces méthodes servent à évaluer la qualité de la solution d'un simulateur telle que le temps de réponse, le taux d'occupation et les longueurs de file d'attente. L'étude de la précision des mesures en phase d'équilibre est présentée dans le chapitre 10 [ED96].

Annexe A

Environnement interactif et graphique de SEDMA

L'interface graphique permet à l'utilisateur de décrire le modèle à simuler, de le valider et d'engendrer automatiquement un code Smalltalk, exécuté par la suite, par le moteur de la simulation dynamique optimiste. L'utilisateur spécifie toutes les opérations du modèle à l'aide de la souris, des manipulations de la souris, du "pop-up" menu et éventuellement des champs du "pop-up" menu. Le résultat de la simulation est ensuite retourné à l'utilisateur dans un fichier ou dans une fenêtre graphique.

L'interface de SEDMA correspond à un langage interactif de description et d'analyse d'un modèle. Son intérêt principal est d'offrir un moyen interactif descriptif pour spécifier le modèle ainsi qu'un moyen visuel interactif pour suivre l'analyse des données caractérisant la performance du système étudié.

L'activité d'un composant actif (sommet) se définit en terme de comportement d'acteur réactif qui spécifie la sémantique associée au sommet. La sémantique est directement programmée interactivement et directement instanciée dans le schéma de comportement de l'acteur.

La spécification du modèle se fait à l'aide du code Smalltalk et l'analyse des résultats se déroulent de manière interactive et visuelle. L'utilisateur devient ainsi un participant actif, capable de lancer la simulation, de suspendre l'exécution, d'examiner, en cours d'exécution, les résultats partiels et de modifier le comportement d'un ou de plusieurs états du modèle simulé.

Le système logique est réalisé à l'aide d'un ensemble d'acteurs réactifs, spécifiques à la simulation par événements discrets répartie optimiste.

Conceptuellement, l'environnement d'interface comporte cinq parties (cf. figure A.1):

- l'éditeur du modèle simulé;
- l'éditeur pour la topologie physique du réseau ou éditeur graphique;
- l'éditeur pour la sémantique du modèle simulé;
- l'environnement de programmation;

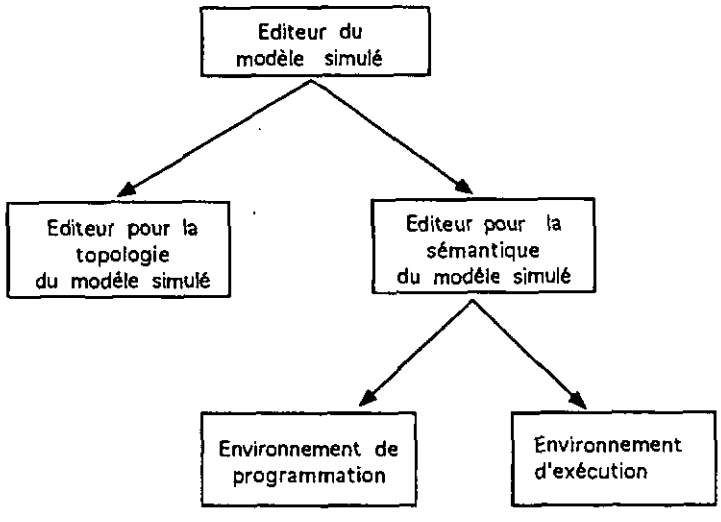


Figure A.1: Environnement interface de SEDMA

- l'environnement d'exécution.

Editeur du modèle simulé

Il rassemble les connaissances du modèle simulé. Le réseau simulé est interactivement défini à partir d'un processeur, puis géographiquement réparti sur les autres processeurs. Cet éditeur, à partir duquel l'ensemble de la simulation est défini et contrôlé interactivement, représente l'éditeur du modèle simulé. Cette partie est reliée aux utilitaires qui permettent de gérer interactivement la structure topologique du réseau qui comporte le grapheur et le système logique qui comporte la sémantique rattachée au grapheur ainsi que le modèle interactif d'exécution.

Cette partie regroupe une quarantaine de classes. Elle comporte des variables d'instance du grapheur et la base de données associée au grapheur.

Editeur pour la topologie du modèle simulé

Cette partie regroupe une soixantaine de classes Smalltalk. Elle rassemble les utilitaires qui servent à spécifier interactivement le réseau rattaché au modèle simulé. Elle comporte les utilitaires pour définir la structure d'un graphe et les algorithmes élémentaires de recherche ainsi que de reconnaissance des composantes fortement connexes dans le graphe du modèle simulé.

SEDMA dispose des classes génériques pour décrire les nœuds de la simulation (par exemple nœud source, nœud de branchement, nœud de diffusion, nœud station-service et nœud puits). Les utilitaires de cette partie servent à manipuler les icônes (les images) associées:

- aux nœuds des composants actifs de la simulation (par exemple nœud source, nœud puits et nœud station-service);

- aux arcs qui établissent les liens entre les différents nœuds de la simulation. Les liens entre les composants sont représentés par des flèches qui indiquent le flux des composants passifs (clients) dans le modèle simulé;

Toutes les manipulations pour définir le graphe du modèle simulé sont rattachées aux **bitmaps** de l'icône.

Editeur du sous-réseau

L'utilisateur peut configurer un ensemble de nœuds en un nœud "subnet node" en utilisant l'éditeur graphique du modèle. Il définit ainsi différents niveaux d'organisation dans le réseau simulé.

La construction des sous-réseaux simulés peut accélérer la modélisation des systèmes en les décomposant en sous-systèmes; ainsi le concepteur et l'utilisateur peuvent-ils visualiser le réseau simulé à plusieurs niveaux d'abstraction.

Les variables qui déterminent le comportement d'un nœud ou d'un sous-réseau peuvent être consultées, examinées et modifiées au cours de l'exécution, grâce aux acteurs-sondes qui font le lien entre l'utilisateur et le noyau de la simulation inaccessible à l'utilisateur.

Editeur pour la sémantique

Cette partie regroupe une trentaine de classes Smalltalk. Elle définit les utilitaires pour la gestion des interactions entre l'environnement d'édition du code Smalltalk de l'utilisateur et son interprétation. Elle sert à interpréter le comportement des acteurs réactifs automatiquement définis et intégrés dans l'environnement de programmation. Cette partie est reliée à l'environnement d'exécution et de programmation interactive du système.

SEDMA dispose des classes génériques pour décrire les codes Smalltalk de chaque type de nœud simulé (par exemple nœud source, nœud de branchement, nœud de diffusion, nœud serveur et nœud puits).

Environnement de programmation

Cette partie regroupe une cinquantaine de classes Smalltalk. Elle constitue l'environnement d'interprétation des comportements pour les acteurs définis à l'aide de l'éditeur de la sémantique. Elle gère une base de données des schémas de comportement et d'accointances. Les schémas de comportement sont représentés en Smalltalk par un ensemble de méthodes et les accointances par un ensemble de sélecteurs. La seule instruction dont l'utilisateur a besoin pour établir la communication entre des acteurs SED est: *receive: aMessageSED.*

Avant de passer à la phase d'exécution, SEDMA vérifie la cohérence du modèle topologique et sémantique du modèle (par exemple le modèle a-t-il au moins une source ou un puits). Les utilitaires de validation vérifient si tous les attributs d'un nœud sont correctement remplis. Ils vérifient également les contraintes soumises à l'utilisateur. Au

cours de cette phase, l'utilisateur se rend compte des violations de contraintes dans le modèle et les corrige.

Environnement d'exécution

Cette partie regroupe environ soixante classes Smalltalk. Elle comporte l'interface de pilotage interactive de la simulation en cours. Les utilitaires mis à disposition permettent de contrôler de manière individuelle l'exécution des acteurs réactifs pour la SED (acteurs logique), des acteurs-sondes, d'examiner les variables d'état des acteurs, de les modifier et de relancer ou de suspendre la simulation.

Le contrôle s'étend d'un acteur quelconque à un ensemble. Le contrôle individuel d'un acteur permet de passer à l'un des modes suivants: terminé, passif ou actif. Dans le mode actif, l'acteur travaille; dans le mode passif, il est suspendu pour être examiné, voire modifié; dans le mode terminé, il a fini son exécution. Le contrôle global se rapporte à l'ensemble du simulateur. Il permet soit de lancer une simulation, soit de faire une pause, soit d'arrêter l'ensemble de la simulation.

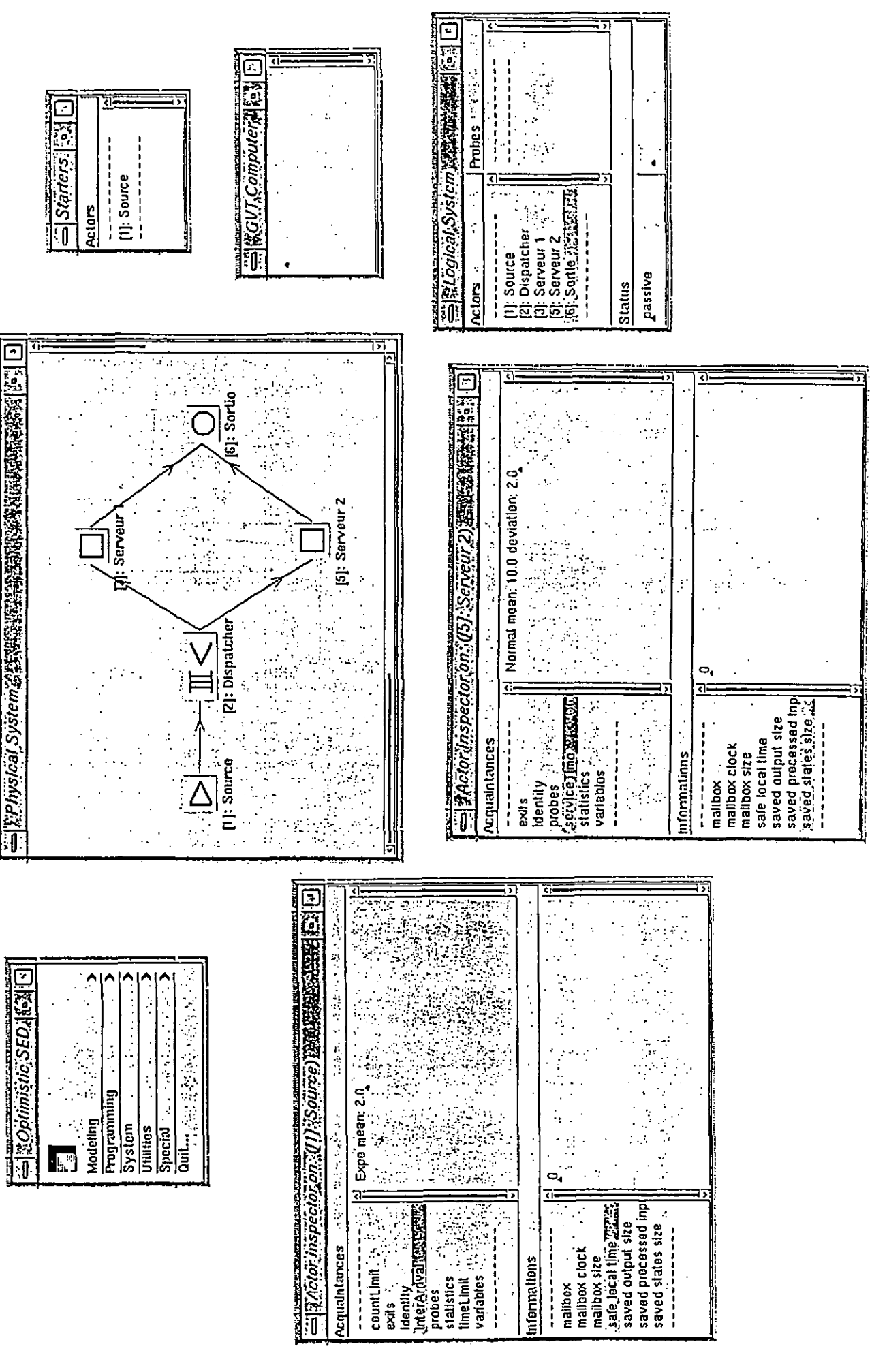
En résumé, les utilitaires d'interface de la simulation peuvent se décrire chronologiquement par:

1. la construction du réseau à l'aide de l'éditeur de graphe; la construction du réseau avec ses différents niveaux organisationnels;
2. l'interface de manipulation directe;
3. l'ajout de la sémantique au réseau; la définition le comportement (les caractéristiques, les lois et les propriétés de chaque composant du modèle simulé);
4. la validation structurelle et sémantique en vue d'engendrer le code automatique;
5. la compilation ou l'interprétation du code Smalltalk qui est par la suite exécuté;
6. l'exécution de la simulation peut être suivie interactivement et visuellement, à tout moment, afin de modifier et d'initialiser les variables statistiques qui s'écartent du résultat;
7. la présentation des résultats statistiques dans un fichier ou dans une fenêtre.

Les éditeurs permettent de construire le réseau simulé (comportant les *PL*s liés entre eux) et de définir le comportement du réseau en lui rattachant la sémantique à l'aide du code Smalltalk.

Annexe B

Spécification du système de lavage de voitures


Optimistic SED
Modeling
Programming
System
Utilities
Special
Quit...
Actor inspector on: ([1]:Source)
Acquaintances
countLimit
exits
identity
interArrivalTime
probes
statistics
timeLimit
variables
Expo mean: 2.0
Informations
mailbox
mailbox clock
mailbox size
safe local time
saved output size
saved processed inp
saved states size
Physical System
[1]: Source
[2]: Dispatcher
[3]: Serveur
[5]: Serveur 2
[6]: Sortio
Starters
Actors
[1]: Source
GUI Computer
Logical System
Actors
Probes
[1]: Source
[2]: Dispatcher
[3]: Serveur 1
[5]: Serveur 2
[6]: Sortie
Status
passive
Actor inspector on: ([5]:Serveur 2)
Acquaintances
exits
identity
probes
serviceTime
statistics
variables
Normal mean: 10.0 deviation: 2.0
Informations
mailbox
mailbox clock
mailbox size
safe local time
saved output size
saved processed inp
saved states size

Annexe C

Analyse des résultats du système de lavage de voiture

Optimistic SED
Metering
Programming
System
Utilities
Special
Quit...
Physical System
[1]: Source
[2]: Dispatcher
[3]: Serveur
[5]: Serveur 2
[6]: Sortie
Actor inspector on: ([1]: A name)
Actor inspector on: ([3]: A name)
Actor inspector on: ([4]: A name)
Actor inspector on: ([5]: A name)
Acquaintances
Informations
exits
identity
interArrival
probes
statistics
timeLimit
variables
mailbox
mailbox clock
mailbox size
clients completed count: 1000
clients interarrival times min: 0.0107497
clients interarrival times max: 102.538
clients interarrival times sum: 15532.1
clients interarrival times mean: 1.0
clients completed count: 606
server service times min: 3.3715
server service times max: 16.3082
server service times sum: 6055.31
server service times mean: 9.99227
server is working: false
BasicQueue 0
clients completed count: 246
server service times min: 15.0411
server service times max: 19.9732
server service times sum: 4320.88
server service times mean: 17.5645
server is working: false
BasicQueue 0
clients completed count: 852
queue length: 148
queue lengths max: 149
queue lengths sum: 142884.0
queue lengths mean: 77.1512
clients waiting times max: 2402.33
clients waiting times sum: 1.0349e6
clients waiting times mean: 1214.67
servers service times min: 3.3715
servers service times max: 19.9732
servers service times sum: 10376.2
servers service times mean: 12.1766
clients completed count: 021
clients existence times min: 11.7011
clients existence times max: 2410.82
clients existence times sum: 1.00260e6
clients existence times mean: 1221.29
clients service times min: 3.3715
clients service times max: 19.9732
clients service times sum: 10080.6
clients service times mean: 12.2784
clients waiting times min: -9.53674e-7
clients waiting times max: 2402.33
clients waiting times sum: 992603.0
clients waiting times mean: 1209.02

Bibliographie

[AD91] H. H. Ammar and S. Deng. Time warp simulation using time scale decomposition. In *Proceedings of the SCS Multiconference on Advances in Parallel and Distributed Simulation*, pages 15–22, 1991.

[Agh86] G. Agha. *Actors: A Model of Concurrent Computation in Distributed Systems.* MIT Press, 1986.

[AR83] A.Goldberg and D. Robson. *Smalltalk-80: the Language and its Implementation.* Addison Wesley, 1983. Series in Computer Science.

[AR92] M. Abrams and P. F. Reynolds Jr., editors. *6th Workshop on Parallel and Distributed Simulation (PADS92). Proceedings of the 1992 SCS Western Simulation Multiconference on Parallel and Distributed Simulation, 20-22 January 1992, Newport Beach, California*, volume 24, no. 3 of *Simulation Series.* SCS, 1992.

[Aya89] R. Ayani. A Parallel Simulation Scheme Based on Distances Between Objects. In [UF89] *Proceedings of the SCS Multiconference on Distributed Simulation*, pages 113–118, 1989.

[BA85] R.A. Blech and D.J. Arpasi. Hardware for a real-time multiprocessor simulator. In *Proceedings of the SCS Multiconference on Distributed Simulation*, volume 15, pages 43–49, January 1985.

[Bau93] H. Bauer. Reducing rollback overhead in time-warp based distributed simulation with optimized incremental state saving. In *Proceeding of the 26th Annual Simulation Symposium*, pages 12–20, April 1993.

[BC93] F. Baccelli and M. Canales. Parallel simulation of stochastic petri nets using recurrence equations. *ACM Transactions on Modeling and Computer Simulation*, 3(1):20–41, January 1993.

[BCC91] R. Baldwin, M.J. Chung, and Y. Chung. Overlapping window algorithm for computing gvt in time warp. In IEEE Computer Society Press, editor, *Proceedings of the 11th International Conference on Distributed Computing Systems*, pages 534–541, May 1991.

[BCLU89] D. Baezner, H. Cleary, G. Lomow, and B. Unger. Algorithmic Optimizations of Simulations on Time Warp. In [UF89] *Proceedings of the SCS Multiconference on Distributed Simulation*, pages 73–78, 1989.

[BCM87] R. L. Bagrodia, K. Many Chandy, and Jayadev Misra. A message-based approach to discrete-event simulation. *IEEE Transactions on Software Engineering*, 13(6), June 1987.

[BDO85] W. Biles, C. Daniels, and T. O'Donnel. Statistical considerations in simulation on a network of microcomputers. In *Proceedings of the Winter Simulation Conference*, pages 388–393, 1985.

[Bel90] S. Bellenot. Global Virtual Time Algorithms. In [Nic90] *Proceedings of the SCS Multiconference on Distributed Simulation*, pages 122–127, 1990.

[Bel92] S. Bellenot. State Skipping Performance with the Time Warp Operating System. In [AR92] *6th Workshop on Parallel and Distributed Simulation (PADS92)*, pages 53–61, 1992.

[BH90] D. Ball and S. Hoyt. The Adaptive Time-Warp Concurrency Control Algorithm. In [Nic90] *Proceedings of the SCS Multiconference on Distributed Simulation*, pages 174–177, 1990.

[BM81] R. S. Boyer and J. S. Moore. *The correctness problem in computer science.* Academic Press, New York, 1981.

[BM90] C. Burdorf and J. Marti. Non-Preemptive Time Warp Scheduling Algorithm. *ACM Operating System Review*, 24(2):7–18, 1990.

[Bou93] T. Bouron. *Structure de communication et d'organisation pour la coopération dans un univers multi-agents.* PhD thesis, Laforia, Paris 6, February 1993.

[BQ92] F. Baccelli and G. Cohen G.J. Olsderand J.-P. Quadrat. *Synchronisation and Linearity: An Algebra for Discrete-Event Systems.* John Wiley, 1992.

[BRF90] C. A. Buzzel, M. J. Robb, and R. M. Fujimoto. Modular vmw rollback hardware for time warp. In [Nic90] *Proceedings of the SCS Multiconference on Distributed Simulation*, pages 153–156, 1990.

[Bri89] J.-P. Briot. Actalk: une plateforme de modèlisation de langages d'acteurs en smalltalk-80. In *7ème Congrès Afcet de Reconnaissance des Formes et Intelligence Artificielle (RFIA'89), Tome 1*, pages 147–161, Novembre-Décembre 1989.

[Bri90] J. V. Briner. *Parallel mixed-level simulation of digital circuits using Virtual Time.* PhD thesis, Duke University, Durham, North Carolina, 1990.

[Bri94] J.-P. Briot. Actalk: une plateforme de modèlisation de langages d'acteurs en smalltalk-80. In *Langages et Modèles à Objets, INRIA/IMAG/PRC-IA, Grenoble, 13-14 Octobre*, 1994.

[Bry88] R. E. Bryant. Data parallel switch-level simulation. In *International Conference on Computer Aided Design (ICCAD'88)*, pages 254–357. IEEE, 1988.

[BS92] H. Bauer and C. Sporrer. Distributed Logic Simulation and an Approach to Asynchronous GVT - Calculation. In [AR92] *6th Workshop on Parallel ond Distributed Simulation (PADS92)*, pages 205–208, 1992.

[CA93] J. Coleman and A. Ambler. Timed automata and discrete event systems. In *Proceedings of the ECC93, Groningen*, July 1993.

[Can94] M. Canales. *Simulation parallèle de réseaux de Pétri Stochastiques: le cas des graphes d'événements.* PhD thesis, Université de Nice-Sophia Antipolis, Fèvrier 1994.

[CG88] J.C. Comfort and R.R. Gopal. Environment partitioned distributed simulation with transputers. In *Proceedings of the SCS Multiconference on Distributed Simulation*, pages 103–108, 1988.

[CG95] V. Camps and M.-P Gleizes. Analysis of altruistic behaviors in multi-agent systems. In *Proceeding of the first International Workshop on Decentralized Intelligent and multi-agents systems; 22-24 navember 1995, Krakov, Poland*, pages II:93–103, November 1995.

[CGU+94] John Cleary, Fabian Gomes, Brian Unger, Xiao Zhonge, and Raimar Thudt. Cost of state saving and rollback. In *Proceedings of the 8th Workshop on Parallel and Distributed Simulation (PADS '94)*, pages 94–101, 1994.

[Cha82] J. Chandy and K. M. Misra. A distributed algorithm for detecting resource deadlocks in distributed systems. In *Proceedings of the ACM SIGACT-SIGOPS Symposion on Principles of Distributes Computing*, pages 157–164, Ottawa, Ontario,Canada, 1982.

[CHM79] K. Chandy, V. Holmes, and J. Misra. Distributed Simulation of Networks. *Computer Networks*, 3:105–113, 1979.

[CK91] A. Concepcion and S. Kelly. Computing Global Virtual Time Using the Multi-Level Token Passing Algorithm. In [MNF91] *Proceedings of the SCS Multiconference on Advances in Parallel and Distributed Simulation*, pages 63–68, 1991.

[CL85] K. M. Chandy and L. Lamport. Distributed snapshots: Determining global states of distributed systems. *ACM Transactions on Computer Systems*, 3(1):63–75, February 1985.

[CM79a] K. Chandy and J. Misra. Distributed Simulation: A Case Study in Design and Verification of Distributed Programs. *IEEE Transactions on Software Engineering*, 5(5):440–452, 1979.

[CM79b] K. M. Chandy and J. Misra. Distributed simulation: A case study in design and verification of distributed programs. *IEEE Transactions on Software Engineering*, SE-5(5):440–452, September 1979.

[CM81] K. Chandy and J. Misra. Asynchronous Distributed Simulation Via a Sequence of Parallel Computations. *CACM*, 24(4):198–206, 1981.

[Coi87] P. Cointe. Metaclasses are firstclass: the objvlisp model. In *OOPSLA'87*, pages 156–167, 1987.

[Con89] A. I. Concepcion. A Hierarchical Computer Architecture for Distributed Simulation. *IEEE Transactions on Computers*, 38(2):311–320, February 1989.

[CS88] B. Cota and R. Sargent. A discrete event simulator with shared memory. In *Proceedings of the SCS Multiconference on Distributed Simulation*, 1988.

[CS89a] K. Chandy and R. Sherman. Space-Time and Simulation. In [UF89] *Proceedings of the SCS Multiconference on Distributed Simulation*, pages 53–57, 1989.

[CS89b] K. M. Chandy and R. Sherman. The conditional event approach to distributed simulation. In [UF89] *Proceedings of the SCS Multiconference on Distributed Simulation*, pages 93–99, 1989.

[CT89] W. Cai and S. J. Turner. An algorithm for distributed discrete-event simulation - the 'carrier null message' approach. In [Nic90] *Proceedings of the SCS Multiconference on Distributed Simulation*, pages 3–8, 1989.

[De 90] R. De Vries. Reducing Null Messages in Misra's Distributed Discrete Event Simulation Method. *IEEE Transactions on Software Enginecring*, 16(1):82–91, 1990.

[DFW94] L. M. D'Souza, X. Fan, and P. A. Wilsey. pGVT: An algorithm for accurate gvt estimation. In *Proceedings of the 8th Workshop on Parallel and Distributed Simulation (PADS '94)*, pages 102–109, 6-8 July 1994.

[DM89] E.H. Durfee and Th. A. Montgomery. Mice: A flexible testbed for intelligent coordination experiments. In *9th International AAAI Workshop on Distributed Artificial Intelligence*, pages 25–40, 1989.

[DMN70] O.J. Dahl, B. Myhrhaug, and K. Nygaard. Simula 67 common base Language. Technical report, Norwegian Computing Centre, Oslo, 1970.

[DR90] P. Dickens and P. Reynolds. SRADS with Local Rollback. In [Nic90] *Proceedings of the SCS Multiconference on Distributed Simulation*, pages 161–164, 1990.

[DS80] E.W. Dijkstra and C.S. Scholten. Termination detection for diffusing computations. *Information Procesing Letters*, 11(1):1–4, August 1980.

[ED96] P.-J. Erard and P. Déguénon. *La simulation par é vénements discrets: concepts et réalisation en Simula, Ada et Smalltalk.* Presses Polytechniques et universitaires romandes, 1996.

[Fer95] A. Ferscha. *Parallel and Distributed Simulation of Discrete Event Systems.* McGraw-Hill, 1995.

[Fil92] J. M. Filloque. *Synchronisation répartie sur une machine parallèle à couche logique reconfigurable.* PhD thesis, Université de Rennes 1, November 1992.

[Fis95] P. A. Fishwick. *Simulation Model Design and Execution: Building Digital Worlds*. Prentice Hall, 1995.

[FR87] R.M. Fujimoto and D.A. Reed. *Multicomputer networks; message based parallel processing*. The MIT Press, 1987. 380 pages.

[FTG88a] R. M. Fujimoto, J. J. Tasai, and G. Gopalakrishnan. Design and performance of special purpose hardware for Time Warp. In *Proceedings of the 15th Annual International Symposium on Computer Architecture*, pages 401–408, 1988.

[FTG88b] Richard M. Fujimoto, Jya-Jang Tsai, and Ganesh Gopalakrishnan. The roll back chip: Hardware support for distributed simulation using time warp. In *Proceedings of the SCS Multiconference on Distributed Simulation*, pages 81–86, February 1988.

[FTG92] R. Fujimoto, J. Tsai, and G. Gopalakrishnan. Design and Evaluation of the Rollback Chip: Special Purpose Hardware for Time Warp. *IEEE Transactions on Computers*, 41(1), January 1992.

[Fuj88] R. Fujimoto. Performance Measurements of Distributed Simulation Strategies. In [UJ88] *Proceedings of the Distributed Simulation Conference*, pages 14–20, 1988.

[Fuj89a] R. Fujimoto. Time Warp on a Shared Memory Multiprocessor. *Trans. Soc. for Comput. Simul.*, 6(3):211–239, July 1989.

[Fuj89b] Richard M. Fujimoto. The virtual time machine. In *International Symposium on Parallel Algorithms and Architectures*, pages 199–208, June 1989.

[Fuj90a] R. Fujimoto. Parallel Discrete Event Simulation. *Comm. of the ACM*, 33(10):30–53, 1990.

[Fuj90b] R. M. Fujimoto. Performance of Time Warp under synthetic workloads. In [Nic90] *Proceedings of the SCS Multiconference on Distributed Simulation*, pages 23–28, 1990.

[Fuj90c] Richard M. Fujimoto. Parallel discrete event simulation. *Communications of the ACM*, 33(10):30–53, October 1990.

[FW95] J. Fleischmann and P. A. Wilsey. Comparative analysis of periodic state saving techniques in time warp simulators. In *Proceedings of the 9th Workshop on Parallel and Distributed Simulation (PADS '95)*, pages 50–58, 1995.

[Gaf88] A. Gafni. Rollback mechanisms for optimistic distributed simulation systems. In [UJ88] *Proceedings of the SCS Multiconference on Distributed Simulation*, pages 61–67, February 1988.

[GD95] Z. Guessoum and P. Déguénon. A multi-agent approch for distributed discrete-event simulation. In *Proceeding of the first International Workshop on Decentralized Intelligent and multi-agents systems; 22-24 november 1995, Krakov, Poland*, pages III:183–190, November 1995.

[GF91] K. Ghosh and R. Fujimoto. Parallel discrete-event simulation using space-time memory. In *Proceedings of the 1991 International Conference on Parallel Processing*, volume 3, pages 201–208, August 1991.

[GGN92] B. Gaujal, A. Greenberg, and D. Nicol. A sweep algorithm for massively parallel simulation of circuit-switched networks. Technical Report 92-30, ICASE, July 1992.

[GH91] W. Glynn and P. Heidelberger. Analysis of Parallel Replicated Simulations Under a Completion Time Constraint. *ACM Transactions on Modeling and Computer Simulation*, 1(1):3–23, 1991.

[GLM91] A. Greenberg, B. Lubachevsky, and I. Mitrani. Algorithms for unboundedly parallel simulations. *ACM Transactions on Computer Systems*, 9(3):541–580, August 1991.

[GT93] D.W. Glazer and C. Tropper. On process migration and load balancing in time warp. *IEEE Transactions on Parallel and Distributed Systems*, 4(3):318–327, March 1993.

[Hei88] P. Heidelberger. Discrete-event simulations and parallel processing: Statistical properties. *SIAM Journal on Statistical Computing*, 9(6):1114–1132, November 1988.

[Hew77] C.E. Hewitt. Viewing control structures as patterns of passing messages. *AI*, 3(8):323–364, 1977.

[Ho92] Y-C. Ho, editor. *Discrete Event Dynamic Systems: Analyzing Complexity and Performance in the Modern World.* IEEE PRESS, 1992.

[Hoa78] C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 23(8):666–677, August 1978.

[IR90] P. Ingels and M. Raynal. Simulation répartie: schéma d'exécution par un modèle à processus. *TSI*, 9(5):383–397, October 1990.

[J+87] D. Jefferson et al. Distributed Simulation and the Time Warp Operating System. *ACM Operating System Review*, 21(1):77–93, 1987.

[JBH+85] D. Jefferson, B. Beckmann, S. Hughes, E. Levy, T. Litwin, J. Spagnuolo, J. Vavrus, F. Wieland, and B. Zimmermann. Implementation of Time Warp on the Caltech Hypercube. In *Proceedings of the International Conference on Distributed Simulation*, pages 70–75, 1985.

[JCRB89] D. Jones, C. Chou, D. Renk, and S. Bruell. Experience with Concurrent Simulation. In *Proceedings of the Winter Simulation Conference*, pages 756–764, 1989.

[Jef85] D. Jefferson. Virtual Time. *ACM Transactions on Programming Languages and Systems*, 7(3):404–425, July 1985.

[Jef90] D. Jefferson. Virtual Time II: Storage Management in Distributed Simulation. Internal Report, UCLA, 1990.

[Jon86] D. Jones. Concurrent Simulation: An Alternative to Distributed Simulation. In *Proceedings of the Winter Simulation Conference*, pages 417–423, Washington, DC, 1986.

[JR91] D. Jefferson and P. Reiher. Supercritical Speedup. In *Annual Simulation Symposium. Record of Proceedings*, pages 159–169, 1991.

[JS85] D. Jefferson and H. Sowizral. Fast Concurrent Simulation Using the Time Warp Mechanism. In *Proceedings of the Conference on Distributed Simulation*, pages 63–69, July 1985.

[Kle75] L. Kleinrock. *Queueing Systems: Theory.* John Wiley and Son, 1975.

[Kok91] P. Kokotovic. *Foundations of Adaptive Control.* Berlin, Heidelberg : Springer-Verlag, 1991.

[Kum89] D. Kumar. An approximate method to predict performance of a distributed simulation scheme. In *Proceedings of the International Conference on Parallel Processing 3*, pages 259–262, August 1989.

[Lam78] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7), July 1978.

[LE81] V. R. Lesser and L. D. Erman. Distributed interpretation: a model and experiment. In *Readings in DAI*, pages 120–139, March 1981.

[Ler80] J. Leroudier. *La simulation à événements discrets.* Hommes et Techniques, 1980.

[Lin92] Y. Lin. Memory Management Algorithms for Optimistic Parallel Simulation. In [AR92] *Proceedings of the 6th Workshop on Parallel and Distributed Simulation (PADS92)*, pages 43–52, 1992.

[Lin94] Yi-Bing Lin. Determining the global progress of parallel simulation with FIFO communication property. *Information Processing Letters*, 50:13–17, 1994.

[Liv85] M. Livny. A study of parallelism in distributed simulation. In [Rey85] *Proceedings of the Conference on Distributed Simulation*, pages 94–98, 1985.

[LK91] A. Law and W.D. Kelton. *Simulation modeling and analysis.* McGraw-Hill, New York, Hamburg, Paris, second edition edition, 1991.

[LL89] Y. Lin and E. Lazowska. The Optimal Checkpoint Interval in Time Warp Parallel Simulation. Technical Report 89-09-04, University of Washington, Seattle, 1989.

[LL90a] Y. Lin and E. Lazowska. Reducing the State Saving Overhead for Time Warp Parallel Simulation. Technical Report 90-02-03, University of Washington, Seattle, 1990.

[LL90b] Y.-B. Lin and E. D. Lazowska. Determining the global virtual time in a distributed simulation. In *Proceedings of the International Conference on Parallel Processing*, pages III–201–III–209, 1990.

[LL91a] Y. Lin and E. Lazowska. Processor scheduling for time warp parallel simulation. In [MNF91] *Proceedings of the SCS Multiconference on Advances in Parallel and Distributed Simulation*, pages 11–14, 1991.

[LL91b] Yi-Bing Lin and Edward D. Lazowska. A time-division algorithm for parallel simulation. *ACM Transactions on Modeling and Computer Simulation*, 1(1):73–83, January 1991.

[LLB90] Y.-B. Lin, E. D. Lazowska, and J.-L. Bauer. Conservative parallel simulation for systems with no lookahead prediction. In *rm [Nic90] Proceedings of the SCS Multiconference on Distributed Simulation*, pages 144–149, 1990.

[LMBS83] S. Lavenberg, R. Muntz, B., and Samadi. Performance analysis of a rollback method for distributed simulation. In A. K. Agrawala and S. K. Tripathi, editors, *Performance'83*, pages 117–132. North Holland, New York, 1983.

[LP91] Y Lin and B. Preiss. Optimal Memory Management for Time Warp Parallel Simulation. *ACM Transactions on Modeling and Computer Simulation*, 1(3):283–307, 1991.

[LPLL93] Yi-Bing Lin, Bruno R. Preiss, Wayne M. Loucks, and Edward D. Lazowska. Selecting the checkpoint interval in time warp simulation. In *Proceedings of the 7th Workshop on Parallel and Distributed Simulation (PADS '93)*, 1993.

[LSW89] B. Lubachevsky, A. Shwartz, and A. Weiss. Rollback Sometimes Works ... If Filtered. In *Proceedings of the 1989 Winter Simulation Conference*, pages 630–639, 1989.

[Lub88] Boris D. Lubachevsky. Bounded lag distributed discrete event simulation. In *Proceedings of the SCS Multiconference on Distributed Simulation*, pages 183–191, 1988.

[Lub89] B. Lubachevsky. Efficient distributed event-driven simulations of multiple-loop networks. *Communications of the ACM*, 32(1):111–123, January 1989.

[LY87] T. Lai and T. Yang. On distributed snapshots. *Information Processing Letters*, 25:153–158, 1987.

[Mat93] Friedemann Mattern. Efficient algorithms for distributed snapshots and global virtual time approximation. *Journal of Parallel and Distributed Computing*, 18(4):423–434, August 1993.

[Meh91] H. Mehl. Speed-Up of Conservative Distributed Discrete Event Simulation Methods by Speculative Computing. In *[MNF91] Proceedings of the SCS Multiconference on Advances in Parallel and Distributed Simulation*, 1991.

[MF94] David M.Nicol and Richard M. Fujimoto. Parallel simulation today. *Annals of Operations Research*, 53:249–285, December 1994.

[Min67] M. Minsky. *Computation: Finite nad infinite machines.* Englewood cliffs, NJ.- Prentice Hall, 1967.

[Mis86] Jayadev Misra. Distributed discrete event simulation. *Computing Surveys*, 18(1), March 1986.

[MMST91] F. Mattern, H. Mehl, A. Schoone, and G. Tel. Global virtual time approximation with distributed termination detection algorithms. Technical Report RUU-CS-91-32, University of Utrecht, 1991.

[MNF91] V. Madisetti, D. Nicol, and R. Fujimoto, editors. *Advances in Parallel and Distributed Simulation, Proceedings of the SCS Multiconference on Advances in Parallel and Distributed Simulation, 23-25 January, 1991, Anaheim, California*, volume 23, no. 1 of *Simulation Series.* SCS, 1991.

[Mol88] P. Moller. *Théorie algébrique des systèmes à événements discrets.* PhD thesis, Ecole Nationale Supérieure des Mines de Paris, Fècembre 1988.

[MT93] Y. Matsumoto and K. Taki. Adaptive time-ceiling for efficient parallel discrete-event simulation. In *Object-Oriented Simulation Conference*, pages 101–106. SCS, January 1993.

[MWM88] V. Madisetti, J. Walrand, and D. Messerschmitt. WOLF: A Rollback Algorithm for Optimistic Distributed Simulation Systems. In *Proceedings of the Winter Simulation Conference*, pages 296–305, San Diego, California, 1988.

[Nev89] C. Nevison. Discrete event simulation using occam. In *Proceedings of the 10th OCCAM User Group*, pages 222–230, 1989.

[Nev90] C. Nevison. Restructuring networks to avoid deadlocks in parallel simulations. In *Proceedings of the Winter Simulation Conference*, pages 446–452, New Orleans, Louisiana, 1990.

[Nic88] D. Nicol. Parallel Discrete-Event Simution of FCFS Stochastic Queueing Networks. *SIGPLAN Notices*, 23(9):124–137, 1988.

[Nic90] D. Nicol, editor. *Distributed Simulation, Proceedings of the SCS Multiconference on Distributed Simulation, 17-19 January, 1990, San Diego, California*, volume 22, no. 1 of *Simulation Series.* SCS, 1990.

[PBH+88] B. Page, R. Boelckow, A. Heymann, R. Kadler, and H. Liebert. *Simulation und moderne Programmiersprachen.* Springer-Verlag, Berlin, Heidelberg, 1988.

[PF89] G. Pujolle and S. Fdida. *Modèles de Systèmes et de Réseaux, Tome 2: Files d'ottente.* = Eyrolles, 1989.

[PLM91] B. Preiss, W. Loucks, and I. Macintyre . Null Message Cancellation in Conservative Distributed Simulation. In [MNF91] *Proceedings of the SCS Multiconference on Advances in Parallel and Distributed Simulation*, pages 33–38, 1991.

[PM89] G.T. Papaspyropoulos and D.G. Maritsas. Parallel discrete event simulation with simula. *Parallel computing*, 12:359–373, November 1989.

[PMW79] J. K. Peacock, E. G. Manning, and J. W. Wong. Distributed simulation using a network of processors. *Computer Networks*, 3(1):44–56, February 1979.

[Pre89] B. Preiss. The Yaddes Distributed Discrete Event Simulation Specification Language and Execution Environment. In [UF89] *Proceedings of the SCS Multiconference on Distributed Simulation*, pages 139–144, 1989.

[PW93a] A. Palaniswamy and P.A. Wilsey. Adaptive bounded time windows in an optimistically synchronized simulator. In *Great Lakes VLSI Conference*, pages 114–118, March 1993.

[PW93b] A. C. Palaniswamy and P. A. Wilsey. An analytical comparison of periodic checkpointing and incremental state saving. In *Proceedings of the 7th Workshop on Parallel and Distributed Simulation (PADS '93)*, 1993.

[RA94] R. Ronngren and R. Ayani. Adaptive checkpointing in time warp. In *Proceedings of the 8th Workshop on Parallel and Distributed Simulation (PADS '94)*, pages 110–117, 1994.

[Ray87] M. Raynal. *Systèmes répartis et réseaux: concepts, outils et algorithmes.* Eyrolles, 1987. (available in the MIT Press, february 1988).

[Rey85] P. Reynolds, editor. *Distributed Simulation, Proceedings of the Conference on Distributed Simulation*, volume 15, no. 2 of *Simulation Series.* SCS, 1985.

[Rey88] P. F. Reynolds Jr. A Spectrum of Options for Parallel Simulation. In *Proceedings of the Winter Simulation Conference*, pages 325–332, San Diego, California, December 1988.

[Rey91] P. F. Reynolds Jr. An Efficient Framework for Parallel Simulations. In [MNF91] *Proceedings of the SCS Multiconference on Advances in Parallel and Distributed Simulation*, pages 167–174, 1991.

[RF87] D.A. Reed and R. Fujimoto. *Multicomputer Networks: Message-Based Parallel Processing.* MIT Press, 1987.

[RMM88] D. Reed, A. Malony, and B. McCredie. Parallel discrete event simulation using shared memory. *IEEE Transactions on Software Engineering*, 14(4):541–553, 1988.

[RST91] M. Raynal, A. Schiper, and S. Toueg. The causal order abstraction and a simple way to implement it. *Information Processing Letters*, 39:343–350, 1991.

[RW89a] R. Righter and J. Walrand. Distributed Simulation of Discrete Event Systems. In *Proceedings of the IEEE*, pages 99–113, 1989.

[RW89b] R. Righter and J.C. Walrand. Distributed simulation of discrete event systems. In *Proceedings of the IEEE*, pages 99–113, January 1989.

[RWJ89] P. Reiher, F. Wieland, and D. Jefferson. Limitation of Optimism in the Time Warp Operating System. In *Proceedings of the 1989 Winter Simulation Conference*, pages 765–770, December 1989.

[Sam85] B. Samadi. *Distributed Simulation: Performance and Analysis.* PhD thesis, Dept. of Computer Science, University of California at Los Angeles, 1985.

[SBW88] L. Sokol, D. P. Briscoe, and A. P. Wieland. Mtw : A strategy for scheduling discrete simulation events for concurrent execution. In [UJ88] *Proceedings of the SCS Multiconference on Distributed Simulation*, pages 34–44, 1988.

[SD80] R.G. Smith and R. Davis. The contract net protocol: Highlevel communication and control in a distributed problem solver. In *IEEE Trans. on Systems, MAn and cybernetics, SMC-11(1)*, pages 1104–1113, November 1980.

[SDC88] S.V. Sheppard, C.K. Davis, and U. Chandra. Parallel simulation environments for multiprocessor architectures. In *Proceedings of the SCS Multiconference on Distributed Simulation, San Diego*, pages 109–114, January 1988.

[SM87] B. Samadi and R. Muntz. A Distributed Algorithm to Detect a Global State of a Distributed Simulation System. In *IFIP'87, Amsterdam*, 1987.

[Sow85] H. A. Sowizral. The time warp simulation system and its performance. In *1985 Society for Computer Simulation Multiconferece*, January 1985.

[SS89] W.-K. Su and C. L. Seitz. Variants of the Chandy-Misra-Bryant Distributed Discrete-Event Simulation Algorithm. In [UF89] *Proceedings of the SCS Multiconference on Distributed Simulation*, pages 38–43, 1989.

[SS90] L. M. Sokol and B. K. Stucky. MTW: experimental results for a constrained optimistic scheduling paradigm. In [Nic90] *Proceedings of the SCS Multiconference on Distributed Simulation*, pages 169–173, 1990.

[Ste91] J. Steinman. Speedes: Synchronous parallel environment for emulation and discrete event simulation. In *[MNF91] Proceedings of the SCS Multiconference on Advances in Parallel and Distributed Simulation*, pages 95–103, 1991.

[Ste93] Jeff S. Steinman. Breathing time warp. In *Proceedings of the 7th Workshop on Parallel and Distributed Simulation (PADS '93)*, Simulation Series, pages 109–118. SCS, May 1993.

[TA89] P. Tinker and R. Agre. Object Creation, Messaging and State Manipulation in an Object Oriented Time Warp System. In [UF89] *Proceedings of the SCS Multiconference on Distributed Simulation*, pages 79–84, 1989.

[Tal93] E.-G. Talbi. *Allocation de processus sur les achitectures parallèles à mémoire distribuée.* PhD thesis, INPG-Grenoble, May 1993.

[Tel91] G. Tel. *Topics in Distributed Algorithms.* Cambridge University Press, England, 1991. 240 pages.

[TG93] A. I. Tomlinson and V. K. Garg. An algorithm for minimally latent global virtual time. In *Proceedings of the 7th Workshop on Parallel and Distributed Simulation (PADS '93)*, 1993.

[TW91] A. Trew and G. Wilson. *Past, Present, Parallel: A Survey of Available Parallel Computing Systems.* Springer-Verlag, 1991.

[TX92] S. Turner and M. XU. Performance Evaluation of the Bounded Time Warp Algorithm. In [AR92] *Proceedings of the 6th Workshop on Parallel and Distributed Simulation (PADS92)*, pages 117–126, 1992.

[UF89] B. Unger and R. Fujimoto, editors. *Distributed Simulation, Proceedings of the SCS Multiconference on Distributed Simulation, 28-31 March, 1989, Tampa,Florida*, volume 21, no. 2 of *Simulation Series.* SCS, 1989.

[UJ88] B. Unger and D. Jefferson, editors. *Distributed Simulation, Proceedings of the SCS Multiconference on Distributed Simulation*, volume 19, no. 2 of *Simulation Series.* SCS, 1988.

[Wes88] D. West. Optimizing time warp: lazy rollback and lazy re-evalution. Master's thesis, University of Calgary, Alberta, 1988.

[WHF+89] F. Wieland, L. Hayley, A. Feinberg, M. DiLoreto, L. Blume, J. Ruffles, P. Reiher, B. Beckmann, P. Hontalas, S. Bellenot, and D. Jefferson. Distributed combat simulation and Time Warp: The model and its performance. In [UF89] *Proceedings of the 1989 SCS Multiconference on Distributed Simulation*, pages 14–20, 1989.

[WP94] P.A. Wilsey and A. Palaniswamy. Rollback relaxation: A technique for reducing rollback costs in an optimistically synchronized simulation. In *International Conference on Simulation and Hardware Description Languages*, pages 143–148. SCS, January 1994.

[Wu90] Y. Wu. *Accélération de la simulation logique: architecture et algorithmes de LL3T.* PhD thesis, INPG Grenoble, September 1990.

[Wya85] D. L. Wyatt. Simulation programming on a distributed system: a preprocessor approach. In *Proceedings of the SCS Multiconference on Distributed Simulation*, pages 32–36, 1985.

[XGUC95] Z. Xiao, F. Gomes, B. Unger, and J. Cleary. A fast asynchronous GVT algorithm for shared memory multiprocessor architectures. In *Proceedings of the 9th Workshop on Parallel and Distributed Simulation (PADS '95)*, pages 203–208, 1995.